Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2016

Hugo dos Santos Frade

Plataforma de Análise Offline da Aplicação de Políticas de Controlo de Acesso Probabilísticas em Bases de Dados Relacionais

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2016

Hugo dos Santos Frade

Plataforma de Análise Offline da Aplicação de Políticas de Controlo de Acesso Probabilísticas em Bases de Dados Relacionais

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Óscar Mortágua Pereira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Dr. António Joaquim da Silva Teixeira
Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Prof. Dr. Marco Paulo Amorim Vieira
Professor Associado com Agregação da Faculdade de Ciência e Tecnologia da Universidade de Coimbra

Prof. Dr. Óscar Mortágua Pereira
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Quero aproveitar para agradecer a todas as pessoas que me apoiaram e ajudaram no meu percurso académico e na realização deste trabalho.

Em especial, expresso o meu profundo agradecimento à minha mãe e à minha namorada pelo apoio dado e por nunca terem deixado de acreditar em mim.

Agradeço também ao professor Óscar Mortágua Pereira, meu orientador, pela sua disponibilidade para me ajudar e guiar neste trabalho.

Resumo

O progresso tecnológico bem como a constante evolução dos diversos modelos de negócio fazem com que os sistemas de informação nos quais estes estão implementados estejam em permanente crescimento. A constante alteração destes sistemas, tanto em termos dos dados efetivamente armazenados bem como dos conjuntos de operações passíveis de serem realizadas torna propício o aparecimento de novas formas de violação das políticas de privacidade estabelecidas, na maioria das vezes inadvertidamente.

Por vezes, através de um conjunto de operações autorizadas ao utilizador é possível a divulgação de informação sensível, através de técnicas de inferência de informação. Perante este cenário, além da utilização de ferramentas que permitam a definição das operações autorizadas e não autorizadas (nas quais estão inseridas a maioria das ferramentas de controlo de acesso), torna-se também necessário recorrer a soluções que ajudem a detetar situações em que os conjuntos de operações legais expõem informação sensível.

Para a resolução deste problema propõe-se o desenvolvimento de uma plataforma que em ambiente de supervisão aceda à informação permitida no âmbito das operações autorizadas e detete estes cenários. Devido à natureza do mesmo, consideramos importante além de detetar quando é possível a um utilizador obter informação sensível, também calcularmos a probabilidade de isso acontecer e, em caso de ela ser superior a um limite predefinido, atuar em conformidade.

Adicionalmente, devido à quantidade de informação com que determinados sistemas operam e, consequentemente, esta ferramenta tem que processar, pretendemos também fornecer flexibilidade ao nível da granularidade com que efetuamos o processo de avaliação dos mecanismos de controlo de acesso implementados, permitindo uma adaptação às necessidades dos arquitetos de sistemas. Posto isto, a granularidade com que o processo de avaliação pode ser efetuado varia entre o nível da célula e o nível do tuplo.

Para uma instância estática da base de dados e, consequentemente circunscritos ao âmbito de operações de leitura, através da identificação das formas como a informação sensível é revelada, a nossa solução ajuda então os arquitetos de sistemas a detetar com precisão a razão das políticas de acesso serem violadas, permitindo-lhes a reconstrução dos casos de uso de forma a evitar que tal aconteça.

Abstract

Technological progress and the constant evolution of the various business models cause its systems to be constantly growing as well. This systems changes, both in terms of stored data as well as the sets of operations capable of being performed makes it conducive the emergence of new ways of violating the established data privacy policies, most often inadvertently.

Occasionally, through a set of allowed operations it is possible for the user to disclose sensitive information, through intormation inference techniques. Taking this into account, besides using frameworks that allow the definition of authorized and not authorized operations (which most of access control frameworks do), it also becomes necessary to use solutions that help to detect situations in which the joint legal operations expose sensitive information.

To solve this problem, we propose the development of a test framework that accesses the allowed information under the authorized operations and expose these scenarios. Due to the nature of it, besides exposing the scenarios where it is possible to disclose sensitive information, this framework must also calculate its probability and act accordingly if it exceeds a predefined limit.

Additionally, due to the amount of information which certain systems operate on and, therefore this tool must process, we intend to provide flexibility in terms of the implemented access control mechanisms evaluation granularity, allowing an adaptation to the system designers needs. Therefore, the granularity with which the evaluation process can be performed varies between the cell level and the tuple level.

For a static instance of the database and, therefore circumscribed to the scope of read operations, through the identification of possible ways of disclosing sensitive information, this solution helps system's architects to pinpoint the reasons why the access policies are violated, allowing them the reconstruction of the use case scenarios to prevent this from happening.

Conteúdo

Lista de Figuras

Lista de Tabelas

Listagens

Glossário

ABAC Attribute-Based Access Control. 17

API Application Programming Interface. 9

CBAC Context-Based Access Control. 17

DAC Discretionary Access Control. 17

DDL Data Definition Language. 11

DML Data Manipulation Language. 12

GC Garbage Collector. 110

IBAC Intent-Based Access Control. 17

JDBC Java Database Connectivity. 34

JVM Java Virtual Machine. 14

LBAC Location-Based Access Control. 17

MAC Mandatory Access Control. 17

PET Privacy Enhancing Technologies. 1

RBAC Role-Based Access Control. 17

REST Representational State Transfer. 14, 15

SGBD Sistemas de Gestão de Bases de Dados. 1

SQL Structured Query Language. 4

Capítulo 1

Introdução

Desde sempre o ser humano necessitou de armazenar e aceder a informação. Com a evolução tecnológica, assistimos a uma migração do armazenamento desta informação do formato em papel para sistemas computacionais. Estes sistemas são comummente suportados por bases de dados, que devem armazenar e organizar a informação de forma a facilitar o acesso à mesma. Por seu lado, as bases de dados são geridas por Sistemas de Gestão de Bases de Dados (SGBD) que são os responsáveis por mediar o acesso à informação propriamente dita.

Neste contexto, os acessos à informação levantam preocupações ao nível da segurança. Entre outras, estas preocupações prendem-se com o fato de saber se um dado sujeito (utilizador do sistema computacional) pode ter acesso a um recurso (informação efetivamente armazenada). Para resolver estes problemas são então formadas políticas de controlo de acesso que previnem o acesso não autorizado a informação sensível.

As políticas de controlo de acesso podem ser especificadas em positivas e negativas. As primeiras formulam os acessos autorizados a um dado sujeito, enquanto que as últimas formulam os acessos negados. Neste contexto são habitualmente desenvolvidas tecnologias de reforço de privacidade, Privacy Enhancing Technologies (PET), que são responsáveis por suportar e aplicar as políticas de de controlo de acesso pretendidas.

No entanto, em certos casos é possível a um utilizador violar as políticas negativas recorrendo a operações que lhe são autorizadas, através de ataques de inferência de informação [1] [2] [3]. Perante este cenário, torna-se então importante que, além de recorrer a ferramentas que apliquem as políticas de controlo de acesso pretendidas, utilizar também soluções que permitam detetar quando, através de um conjunto de operações autorizadas é possível inferir informação que devia ser protegida. Através da utilização das últimas, torna-se então possível aos arquitetos de sistemas alterar os conjuntos de operações autorizadas, ainda na fase de desenvolvimento do mesmo.

A solução desenvolvida nesta dissertação corresponde então a uma plataforma que permite avaliar o modelo de controlo de acesso implementado, verificando se é possível, através de operações autorizadas, inferir informação sensível. As políticas de controlo de acesso definem a informação à qual um utilizador pode ou não aceder. No âmbito desta solução pretende-se desenvolver um sistema que além de permitir detetar quando um utilizador consegue inferir informação à qual não deve ter acesso, consiga também identificar a probabilidade desse acontecimento e alertar o respetivo programador da aplicação quando a mesma é superior a um determinado limite estabelecido.

Por fim, pretende-se que esta verificação possa ser efetuada com diferentes graus de gra-

nularidade. A granularidade da avaliação realizada é tipicamente efetuada ao nível da célula, podendo no limite, ser efetuada ao nível do tuplo.

1.1 Motivação

As soluções de controlo de acesso servem para evitar que um utilizador tenha acesso a informação considerada sensível. Estas soluções assentam na definição de políticas de controlo de acesso que definem claramente qual a informação que pode ou não ser acedida por cada utilizador. Por exemplo, no âmbito de uma empresa, cujo esquema da base de dados é apresentado na figura A.1 do Anexo `A`, um cliente não pode ter acesso à informação de compras que não tenham sido efetuadas por si. Nesta situação sempre que o utilizador (cliente) tenta aceder a uma lista de compras, a mesma pode ser filtrada pela condição (customerID = getUserID()).

No entanto, por vezes é desejável que um utilizador tenha acesso a informação *sensível*. A título de exemplo, no contexto da empresa supramencionada, considere-se a política de controlo de acesso em que um utilizador não pode ter acesso à informação de um empregado, caso o mesmo tenha uma salário superior a S_1. Neste caso, ao procurar pela lista de empregados na base de dados da empresa, é desejável que o utilizador tenha acesso, por exemplo, a EmployeeID, LastName e FirstName de todos os empregados. Perante este cenário, apenas se considera quebra da política de controlo de acesso estabelecida quando um utilizador consegue associar um empregado a um salário superior a S_1. De seguida será explorado este exemplo.

Tal como já referido, existem soluções que permitem a definição de políticas de controlo de acesso e a aplicação das mesmas. No entanto, perante certos cenários é possível através da execução de operações autorizadas inferir informação que de outra forma não deveria ser divulgada.

Para a base de dados da empresa supramencionada, que possui uma relação Employees, cujo estado é apresentado na tabela 1.1, suponha-se que o administrador da mesma pretende implementar uma política de controlo de acesso que previna um utilizador saber quais os empregados que possuem um salário superior a 2000.

EmployeeID	LastName	FirstName	Country	Salary
1	Davolio	Nancy	USA	2954.55
2	Fuller	Andrew	USA	2254.49
3	Peacock	Margaret	USA	1861.08
4	Leverling	Janet	USA	3119.15
5	Buchanan	Steven	UK	1744.21
6	Suyama	Michael	UK	2004.07
7	King	Robert	UK	1991.55
8	Callahan	Laura	USA	2100.5
9	Dodsworth	Anne	UK	2333.33

Tabela 1.1: Exemplo da tabela Employees

Adicionalmente, ao utilizador é permitida a execução das queries apresentadas na listagem

1.1.

```
select EmployeeID, FirstName, LastName from Employees;
select EmployeeID, FirstName, LastName from Employees where Salary < 2000;
```

Listagem 1.1: Definição das queries autorizadas

Neste cenário, apesar de não se poder executar uma query que retorne diretamente os empregados que possuem um salário superior a 2000, pode-se, através das queries apresentadas inferir esta informação. Tendo em conta o estado da tabela 1.1, pode-se constatar que a diferença entre os resultados obtidos no âmbito da execução da primeira e da segunda queries é apresentada na tabela1.2.

EmployeeID	LastName	FirstName
1	Davolio	Nancy
2	Fuller	Andrew
4	Leverling	Janet
6	Suyama	Michael
8	Callahan	Laura
9	Dodsworth	Anne

Tabela 1.2: Diferença entre os resultados obtidos na primeira e segunda queries

Perante estes resultados, é possível perceber que através de uma operação de subtração entre o conjunto de tuplos retornados pela primeira e segunda queries é possível a um utilizador inferir quais os empregados que possuem um salário superior a 2000, violando assim a política de controlo de acesso definida.

Adicionalmente, no âmbito desta abordagem de controlo de acesso, onde é permitida a obtenção de informação sensível (informação dos empregados) e apenas se considera que ocorre violação das políticas estabelecidas no caso de se conseguir associá-la a valores críticos (salários superiores a 2000), surge um problema adicional; mesmo quando o utilizador não sabe efetivamente quais os empregados que possuem um salário superior a 2000, não se pode ignorar o fato de o mesmo ter acedido a informação sobre os mesmos. No exemplo apresentado, foi possível perceber que através da execução da primeira query o utilizador acedeu a informação de 9 empregados, dos quais 6 possuíam um salário superior a 2000. Nesta situação, a probabilidade de o utilizador acertar cegamente num empregado com um salário superior a 2000 é de 66.66%. Igualmente, pôde-se também verificar que após a execução da segunda query, momento a partir do qual é possível um utilizador perceber quais os empregados que correspondem a informação sensível, esta probabilidade passa a ser de 100%. Neste contexto compreendeu-se a necessidade de controlar a probabilidade, no âmbito da informação já obtida, de um utilizador conseguir acertar cegamente em informação sensível.

Atentando agora num novo caso em que o estado da base de dados corresponde ao apresentado na tabela 1.1 e que o administrador da mesma pretende implementar uma política de controlo de acesso que previna um utilizador de saber quais os empregados que possuem um salário superior a 2500. Suponha-se que são submetidas as queries q_1, q_2 e q_3.

```
q1: select EmployeeID, LastName, FirstName from Employees where EmployeeID < 4;
```

```
q2: select Salary from Employees where EmployeeID = 2;
q3: select EmployeeID, LastName, FirstName from Employees where EmployeeID >= 4;
```

Listagem 1.2: Definição das queries autorizadas

Primeiramente o utilizador obtém a informação relativa a 3 empregados sendo que apenas o empregado identificado por (EmployeeID = 1) possui um salário superior a 2500. Ou seja, após a execução da primeira query tem uma probabilidade de 33.33% de acertar na identificação do empregado supramencionado como tendo um um salário superior a 2500. Ao executar a segunda query fica a saber o salário do empregado identificado por (EmployeeID = 2), e que o mesmo é inferior a 2500. Posto isto, esta probabilidade passa a ser de 50%. Após a execução da terceira query acaba por obter a informação relativa a todos os empregados restantes. Ou seja, tendo em conta o estado da base de dados, tem uma probabilidade de 25% de *acertar* ao identificar um empregado como tendo um um salário superior a 2500.

Perante este caso, onde, à medida que o utilizador executa sequencialmente as queries e obtém mais informação, pode-se constatar que a probabilidade de acerto na identificação de informação sensível já obtida varia. Posto isto, em certas situações (que devem ser identificadas pelo administrador) torna-se necessária a avaliação da informação passível de ser inferida no fim da execução de cada query.

Tal como nos exemplos apresentados, casos em que é possível inferir informação sensível a partir de operações autorizadas, levam à necessidade do desenvolvimento de soluções que permitam detetá-los.

1.2 Solução Proposta

A solução proposta no âmbito desta dissertação pretende resolver o problema da inferência de informação ao nível de declarações Structured Query Language (SQL) de consulta a bases de dados relacionais. Foi concebida para ser utilizada com o propósito de detetar se através de conjuntos de operações autorizadas, é possível inferir informação sensível. O fato de esta verificação não poder ser efetuada em tempo de execução da aplicação deve-se:

1. ao processamento necessário efetuar;
2. ao fato de por vezes, pelo simples fato de se negar o acesso a informação, o utilizador poder deduzir que se trata de informação sensível.

Neste contexto, ao nível da simplificação do processamento necessário de efetuar e, consequentemente, do tempo requerido, pretende-se que o processo de avaliação possa ser efetuado com diferentes níveis de granularidade. Posto isto, às custas de uma granularidade de grau mais elevado (ou seja, não sendo tão preciso no processo de avaliação), será possível obter um processamento mais rápido.

Adicionalmente, o fato de esta avaliação ser efetuada em tempo de execução, logo estando sujeita a alterações ao nível do esquema da base de dados, bem como ao nível da informação efetivamente armazenada, implicaria mecanismos adicionais que permitissem responder a estes tipos de atualizações.

Esta solução recorre a uma base de dados auxiliar (bd_{aux}), inicializada com toda a informação armazenada na base de dados relacional que se pretende proteger. Posteriormente, a informação obtida por cada query submetida pelo administrador é também armazenada em

bd_{aux}, sendo sobre esta informação que a plataforma determina a informação passível de ser inferida por um utilizador malicioso. A informação com a qual bd_{aux} é inicializada tem como propósito a validação da informação inferida a cada passo no processo de avaliação, tal como se poderá perceber no capítulo 4.

O objetivo da mesma passa por, aquando da deteção da violação das políticas de controlo de acesso estabelecidas, produzir um relatório que possa auxiliar o administrador/*developer* a identificar quais as operações autorizadas que levam ao acesso indevido da informação protegida.

De fato, tal como já mencionado, além de se pretender prevenir que um utilizador consiga perceber, de entre a informação que obteve, qual corresponde efetivamente a informação sensível, é necessário também ter em conta a probabilidade do mesmo acertar cegamente na mesma. Neste contexto surge o conceito de política de controlo de acesso probabilística. Um política de controlo de acesso probabilística é definida sob a forma de tuplo

<política de controlo de acesso> <probabilidade limite de acerto>

em que a componente <probabilidade limite de acerto> representa o limite de probabilidade de acerto acima do qual é considerado que existe uma quebra da política de controlo de acesso. Neste contexto, pode então ser definida uma <probabilidade limite de acerto> para cada <política de controlo de acesso> que se pretenda implementar.

Posto isto, o caso limite de <probabilidade limite de acerto> ser 0%, corresponde a um cenário em que se pretende que um utilizador não possa aceder a informação sensível nenhuma, independentemente de a conseguir identificar como tal. Por outro lado, o fato de a componente <probabilidade limite de acerto> tomar o valor de 100% simboliza o fato de apenas se pretender evitar que um utilizador consiga identificar com certeza que determinada informação corresponde a informação sensível.

Tendo em conta o esquema da base de dados apresentado na figura A.1, apresentada no Anexo `A`, outros exemplos de cenários que se pretendem analisar na solução desenvolvida nesta dissertação podem ser:

- impedir que um dado utilizador saiba que `Customers` efetuaram compras de produtos cujo preço seja superior a um determinado valor (v_1);
- impedir que um dado utilizador saiba que `Suppliers` fornecem produtos de determinada categoria.

Convém relembrar que a violação destas políticas de controlo de acesso não corresponde ao acesso à informação relativa aos Customers, valores dos produtos, Suppliers e categorias de produtos de uma empresa mas sim à associação entre os mesmos, quando os valores/categorias dos produtos são *inválidos*. De fato, não é necessário que o utilizador saiba quais os valores efetivos dos valores/categorias dos produtos para que haja violação das políticas de controlo de acesso, tal como ficou patente nos exemplos apresentados no âmbito das listagens 1.1 e 1.2.

Tal como referido na seção 1.1, no desenvolvimento desta solução pretende-se fornecer ao administrador a flexibilidade de definir os momentos em que pretende que o processo de avaliação do modelo de controlo de acesso implementado seja efetuado. Sendo assim, em cenários mais otimistas, onde se parte do princípio em que o modelo de controlo de acesso implementado previne a inferência de informação sensível, é possível executar o processo de

avaliação apenas no fim do caso de uso. Alternativamente, em cenários mais pessimistas, o administrador pode definir as fases do caso de uso em que o modelo de controlo de acesso implementado é avaliado, permitindo assim uma identificação mais precisa das operações que permitem a violação das políticas de controlo de acesso estabelecidas.

1.3 Ferramentas utilizadas

A linguagem de programação utilizada no âmbito desta dissertação corresponde à versão 8 do Java[4], visto que fornece um elevado grau de portabilidade às aplicações desenvolvidas na mesma. Um motivo adicional para a utilização desta linguagem prende-se com o fato de as ferramentas utilizadas no âmbito do desenvolvimento desta solução possuírem uma implementação em Java, o que permite a utilização das mesmas em modo embutido. Foi desenvolvida no Netbeans IDE[5], que permite uma fácil organização dos componentes deste trabalho, facilitando-o.

Primeiramente, no âmbito do desenvolvimento desta dissertação, foi necessário recorrer a um analisador de queries SQL para se poder interpretar as queries submetidas. Neste contexto, de entre as possíveis ferramentas disponíveis, tais como FoundationDB[6] e General SQL Parser[7], optou-se por recorrer à ferramenta FoundationDB. O motivo da escolha desta solução prende-se com o fato de a mesma ser grátis (à altura do desenvolvimento deste componente da dissertação) e relativamente simples de implementar, utilizando um *visitor design pattern*[8].

Adicionalmente, tal como referido na seção 1.2, foi utilizada uma base de dados auxiliar. O problema que se pretendeu resolver no âmbito desta dissertação, foi solucionado parcialmente numa primeira abordagem recorrendo ao paradigma de grafos. Como se poderá constatar, esta solução possuía algumas limitações o que levou a que fosse necessário o desenvolvimento de uma segunda abordagem, recorrendo ao paradigma de hipergrafos.

No âmbito da primeira solução, de entre as bases de dados de grafos existentes, como por exemplo Neo4j[9], Titan[10], AllegroGraph[11], GraphBase[12], OrientDB[13], GraphDB[14] e FlockDB[15], optou-se por recorrer à base de dados de grafos Neo4j. A escolha recaiu sobre esta solução uma vez que a mesma corresponde a uma implementação de uma base de dados de grafos (grátis) em Java, o que permite que seja utilizada em modo embutido. Esta caraterística faz com que a mesma seja fácil de aprender e utilizar, apresentando ao mesmo tempo um elevado desempenho.

Relativamente à segunda solução, que corresponde à solução final desta dissertação, a escolha recaiu sobre a base de dados HypergraphDB[16]. O motivo desta escolha prendeu-se novamente com o fato de esta corresponder a uma solução implementada em Java (permitindo assim a sua utilização em modo embutido), *open-source* e devido à complexidade do modelo de dados que a mesma permite implementar. Como se poderá perceber no capítulo 4, foi necessário recorrer a coleções suportadas em disco. Devido à forma como HypergraphDB modela a informação armazenada, é possível, ao nível de uma instância da mesma desenvolver estas coleções, o que permite eliminar o *overhead* do processo de obtenção de referências para os objetos armazenados na base de dados.

A base de dados relacional utilizada no âmbito da prova de conceito desta solução foi o MySQL[17].

1.4 Organização da Dissertação

O documento está organizado da seguinte forma:

- O capítulo 2 apresenta o `Estado da Arte e Conhecimento Prévio`, onde é apresentado o modelo relacional, o paradigma da teoria de grafos, os tipos de políticas de controlo de acesso existentes e alguns exemplos de soluções que as aplicam. Adicionalmente, serão apresentadas também algumas tecnologias existentes utilizadas no âmbito desta dissertação;
- O capítulo 3 apresenta a `Solução concetual`, onde são descritas as políticas de controlo de acesso probabilísticas, o paradigma utilizado no âmbito do armazenamento de informação e do processo de avaliação;
- O capítulo 4 apresenta a `Prova de Conceito`, onde é abordado o desenvolvimento da solução previamente descrita: começa-se por apresentar a arquitetura geral da solução desenvolvida e, de seguida, para cada um dos seus componentes, são explicados os detalhes e as estratégias utilizados na sua implementação;
- O capítulo 5 apresenta a `Conclusão`, onde é discutido o trabalho realizado no âmbito desta dissertação bem como o trabalho que pode ser feito para a completar.

Capítulo 2

Estado da Arte e Conhecimento Prévio

Neste capítulo será apresentado o estado da arte dos diversos tópicos abrangidos por esta dissertação. Primeiramente será apresentado o `Modelo Relacional` na seção 2.1 e os diversos `Sistemas de Gestão de Bases de Dados Relacionais` em 2.1.1, bem como a linguagem utilizada para aceder aos mesmos em 2.1.2.

De seguida, será exposto `Modelo de Grafos` na seção 2.2, as abordagens de busca recursiva em grafos em 2.2.1 e, por fim, algumas implementações existentes de bases de dados de grafos.

Posteriormente, será descrito `Controlo de Acesso` na seção 2.3, onde são apresentados os principais modelos de controlo de acesso em 2.3.1 e os mecanismos de controlo de acesso de baixo nível em 2.3.3. Por fim, no âmbito do conceito de controlo de acesso, serão expostas algumas soluções existentes que aplicam o controlo de acesso em bases de dados relacionais em 2.3.4.

Por fim, no âmbito da seçao 2.4 será apresentada a linguagem Java bem como a Application Programming Interface (API) JDBC em 2.4.1.

2.1 Modelo Relacional

O Modelo Relacional, inicialmente apresentado em [18], corresponde a uma abordagem a um modelo matemático baseado na teoria de conjuntos para gerir dados de forma organizada e estruturada. Neste modelo, a principal forma de organização de dados denomina-se por `relação`. Uma relação é definida por um esquema - composto pelo nome da relação e pelo nome e domínio das colunas que a compõem, também denominadas de atributos - e por uma instância do mesmo. A instância de uma relação corresponde ao conjunto de linhas armazenadas em cada momento, também denominadas de `tuplos`, correspondendo estes ao ponto básico de correlação de informação.

Como se pode constatar constatar, a tabela 2.1 armazena informação relativa a compras de uma determinada empresa, sendo o esquema da relação definido por *Orders(OrderID: long, CustomerID: string, EmployeeID: int, OrderDate: date)*.

Cada tuplo de uma relação é diferente de todos os outros, correspondendo ao seu identificador único uma `chave primária`. As chaves primárias correspondem ao conjunto de um ou mais atributos de uma relação que identificam unicamente os seus tuplos. Além deste

OrderID	CustomerID	EmployeeID	OrderDate
10248	VINET	1	1996-07-04
10249	TOMSP	2	1996-07-0

Tabela 2.1: Exemplo de Tabela Orders

tipo de chaves, existem também as denominadas `chaves estrangeiras`, que correspondem tipicamente a referências para chaves primárias de outras tabelas. É através destas chaves estrangeiras que é possível associar tuplos de múltiplas tabelas, correlacionando assim os múltiplos domínios de informação existentes que foram transportados do mundo real.

Foram definidas algumas regras que visam garantir a integridade dos dados:

- `Domínio` dos atributos. Forma mais elementar de integridade; os campos devem obedecer ao tipo de dados e às restrições de valores admitidos por um atributo;
- `Entidade` de cada tuplo deve ser identificado de forma única com recurso a uma chave primária que não se repete e não pode ser `null` (condição de conjunto);
- `Referencial` o valor de uma chave estrangeira ou é `null` ou contém um valor que é chave primária na relação de onde foi importada.

Recorrendo ao exemplo supramencionado, uma compra possui um identificador único (*OrderID*), efetuada por um cliente (*CustomerID*) e a um empregado (*EmployeeID*). Os atributos *CustomerID* e *EmployeeID* correspondem a chaves estrangeiras desta relação servindo para identificar tanto o cliente como o empregado que participaram na ação da compra.

Como referido em [18], uma relação (tabela) corresponde a um conjunto. Assim, todas as operações que são passíveis de serem aplicadas sobre conjuntos, podem igualmente ser aplicadas sobre relações de uma base de dados relacional.

Neste âmbito, ao nível das operações de leitura, é então possível obter a `projeção` (subconjunto) de um determinado conjunto de atributos de uma relação. Formalizando esta descrição, se uma lista `L` de `k` índices $L = i_1, i_2, ..., i_k$ e `R` um n-ário tal que $n \geq k$, então a projeção $\pi_L(R)$ corresponde ao k-ário, cujo atributo de índice j corresponde ao atributo i_j de `R`, sendo j =1, 2, ..., k. A título de exemplo, para a relação Orders apresentada na tabela 2.1, cujo esquema já foi apresentado, é possível obter o subconjunto de atributos desta relação definido por π_{Orders} = { CustomerID, EmployeeiD, OrderDate}.

EmployeeID	LastName	FirstName
1	Davolio	Nancy
2	Fuller	Andrew

Tabela 2.2: Exemplo de tabela Employees

Adicionalmente é também possível realizar `composição` de conjuntos de atributos que têm pelo menos um atributo em comum. Para as relações R e S, se existirem duas projeções tais

que $\pi_1(R) = \pi_2(S)$, então é possível fazer a composição de dois conjuntos. Recorrendo às tabelas *Orders*, apresentada em 2.1 e *Employees* apresentada em 2.2, é possível fazer a união dos dois conjuntos se usar uma projeção em *EmployeeID* em ambas as relações, resultando assim um novo conjunto, apresentado na tabela 2.3.

EmployeeID	LastName	FirstName	OrderID	CustomerID	OrderDate
1	Davolio	Nancy	10248	VINET	1996-07-04 00:00:00.00
2	Fuller	Andrew	10249	TOMSP	1996-07-05 00:00:00.00

Tabela 2.3: Exemplo da União da tabela Orders com Employees

2.1.1 Sistemas de Gestão de Bases de Dados Relacionais

Os SGBD são sistemas que facilitam o processo de Definição, Construção, Manipulação e Partilha de bases de dados entre vários utilizadores e/ou aplicações.

- `Definição` corresponde ao processo de especificação do tipo de dados, estruturas de dados e restrições;
- `Construção` corresponde ao processo de armazenamento de dados;
- `Manipulação` envolve operações como a pesquisa e obtenção de dados;
- `Partilha` corresponde ao mecanismo que gere o acesso (simultâneo) aos dados por parte de vários utilizadores e programas.

No âmbito dos requisitos funcionais destes sistemas, são eles que também estão encarregues da independência entre programas e dados, integridade dos dados (controlo de alteração de dados de acordo com as regras de integridade), consistência dos dados (nos processos de transações e mesmo em falhas de software/hardware), eficiência no acesso aos dados (especialmente em cenários de manipulação de grandes quantidades de dados), isolamento de utilizadores (cada utilizador tem a "sensação" de ser o único), melhor gestão do acesso concorrencial, serviços de segurança (controlo de acessos/permissões), codificação de dados (mecanismos backup e de recuperação de dados), administração de dados e linguagem de desenho e manipulação de dados.

Alguns dos exemplos de SGBDs são `Oracle Database`, `Microsoft SQL Server`, `MySQL`, entre outros.

2.1.2 Linguagem de Query Estruturada

O SQL é uma linguagem usada para definir, manipular e questionar uma base de dados relacional. É orientada ao processamento de conjuntos. Possui duas sub-linguagens principais:

- Data Definition Language (DDL), que é utilizada para definir as várias relações; o esquema de cada uma, as restrições de integridade, conjunto de índices, entre outras operações;

- Data Manipulation Language (DML), que é utilizada para inserir, eliminar, atualizar e consultar dados.

Neste âmbito, dando especial atenção às declarações de consulta (queries) realizadas através da linguagem SQL, estas correspondem tipicamente a operações sobre conjuntos e de álgebra relacional, tal como foi mencionado no início desta secção. À semelhança de na `Teoria de Conjuntos` onde não existe a ocorrência de múltiplos elementos iguais, também esta linguagem permite lidar, aquando de uma pesquisa de um subconjunto dos atributos de uma relação, com tuplos de informação duplicada, tendo comandos próprios para eliminar os duplicados.

Além de ser possível pesquisar apenas subconjuntos de atributos numa relação, é também possível filtrar os tuplos retornados da mesma recorrendo a expressões condicionais. Nesta linguagem, a utilização destes *filtros* nas consultas está tipicamente presente nas cláusulas where das queries sql. Um exemplo deste tipo de queries é apresentado na listagem 2.1.

```
1 select * from Employees where EmployeeID <= 100;
```

Listagem 2.1: Exemplo de query com cláusula where

Por fim, e como foi apresentado no início da seção 2.1, a linguagem SQL possibilita também que se consulte informação de mais do que uma relação numa mesma query. Esta instrução é executada tipicamente recorrendo a à cláusula `join`, sendo o resultado da query 2.2, o exposto na listagem 2.3

```
1 select Employees.EmployeeID as EmployeeID, LastName, FirstName, OrderID,
    CustomerID, OrderDate
2 from Employees join Orders
3 on Employees.EmployeeID = Orders.EmployeeID;
```

Listagem 2.2: Consulta de múltiplas tabelas

2.2 Modelo de Grafos

Tal como mencionado em [19], um grafo pode ser visto como uma rede de nós/vértices conetados por arcos/arestas. Apresentando este conceito de forma formal, um Grafo G consiste num conjunto V_G de vértices e um conjunto E_G de arestas. Cada aresta e contém um conjunto $V_G(e)$ de nós que é composto por um ou dois elementos do conjunto de V_G. As arestas, definidas podem ser direcionais (digrafos), quando possuem sentido, podendo ser atravessadas apenas do nó inicial para o nó final, ou bidirecionais, podendo ser atravessadas a partir de qualquer um dos nós que interligam. Neste contexto surge o conceito de incidência; o conjunto de incidência de um vértice v_1 corresponde ao conjunto de arestas que começam/acabam em v_1.

Posto isto, para um conjunto $V_G = \{v_1, v_2, v_3, v_4, v_5\}$, $E_G = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, sendo $V_G(e_1) = \{v_1, v_2\}$, $V_G(e_2) = \{v_1, v_3\}$, $V_G(e_3) = \{v_2, v_3\}$, $V_G(e_4) = \{v_3, v_4\}$, $V_G(e_5) = \{v_4, v_5\}$ e $V_G(e_6) = \{v_1, v_5\}$, a sua representação topológica corresponde à apresentada na figura 2.1.

Este tipo de representação é propício a operações que envolvam descoberta de relações entre valores de diferentes nós, onde os grafos são atravessados através de `caminhos`; conceitos de `vizinhança` e `conetividade`.

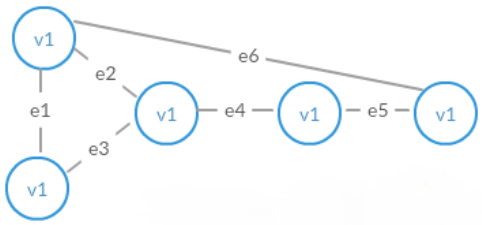

Figura 2.1: Exemplo da representação de um grafo

De fato, o conceito de grafo pode ser considerado como um caso particular do conceito de hipergrafo [20][21], cuja representação pode ser vista como a apresentada na figura 2.2.

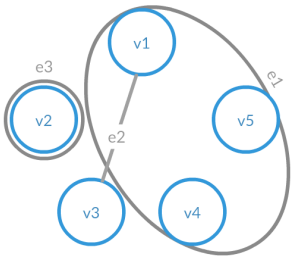

Figura 2.2: Exemplo da representação de um hipergrafo

As arestas são representadas por

- uma linha a conetar os vértices, caso a respetiva cardinalidade seja 2;
- um círculo fechado que inclui os vértices, caso a respetiva cardinalidade seja diferente de 2.

Tal como apresentado em [22], apesar de as bases de dados de grafos terem surgido nos anos 80, a sua relevância começou a surgir recentemente com a necessidade de manipular informação, cuja representação natural se assemelhe à representação de grafos. Ou seja, quando a informação sobre as relações e a topologia do esquema passou a ser tão ou mais importante do que os próprios dados armazenados.

As aplicações mais comuns deste tipo de bases de dados, correspondem a redes sociais, onde é importante armazenar as relações (arestas) entre múltiplos utilizadores (nós); redes tecnológicas, arquiteturas de redes, redes de poder energético, redes de rotas aéreas, entre outros exemplos, em que os aspetos geográficos e espaciais são relevantes; modelos de redes de informação que representem o *flow* das mesmas, tal como citações entre *papers* académicos; entre outros.

De seguida será apresentado o mecanismo de procura exaustiva em grafos em 2.2.1 e por fim serão apresentadas as soluções de bases de dados de grafos e hipergrafos utilizadas nesta dissertação nas subseções 2.2.2 e 2.2.3, respetivamente.

2.2.1 Abordagens de busca exaustiva em grafos

Quando se recorre a teoria de grafos para solucionar um problema, tipicamente pretende-se fazer uso de um mecanismo de busca de (pelo menos) um vértice a partir de um conjunto

de vértices. Esta pesquisa tipicamente necessita de ser exaustiva. Neste contexto é possível utilizar duas estratégias: `busca em profundidade` e `busca em largura` [23].

A primeira começa por tentar obter o vértice pretendido (v_{dest}) através da expansão do primeiro vértice adjacente ao vértice de origem (v), aprofundando-se cada vez mais até que encontre o v_{dest} ou que detete que não tem mais vértices *novos* para atravessar. Enquanto a segunda hipótese se verificar a busca retrocede (*backtrack*), recomeçando no nó pai que ainda não foi expandido. Recorrendo a uma estratégia não recursiva, esta abordagem é implementada recorrendo a uma pilha[1], tal como é apresentado no algoritmo 1.

```
1 let stack = (v);
2 while stack is not empty do
3 |   x = top(stack);
4 |   if x is adjacent to a new vertex y then  add y to the top of stack ;
5 |   else remove x from stack;
6 end
```

Algoritmo 1: Algoritmo de busca em profundidade [23]

No âmbito da `busca em largura` a cada iteração verificam-se todos os vértices adjacentes ao conjunto de vértices corrente antes de avançar para a próxima iteração. Posto isto, não é necessário *backtracking*. O algoritmo 2 apresenta a forma como se pode implementar uma solução de `busca em largura`. Neste contexto, pode-se constatar que a mesma recorre a uma fila[2] em vez de a uma pilha.

```
1 let queue = (v);
2 while queue is not empty do
3 |   x = front(stack);
4 |   if x is adjacent to a new vertex y then add y to the top of queue;
5 |   else remove x from queue;
6 end
```

Algoritmo 2: Algoritmo de busca em largura [23]

2.2.2 Neo4j

A base de dados de grafos Neo4j[9], implementa uma estrutura em grafo tradicional, contendo objetos que representam as duas entidades básicas neste paradigma: nós e arestas, sendo atribuídos valores a estas entidades, por forma a poderem ser identificadas. Todas as operações que alteram os dados na base de dados utilizam transações por forma a garantir a integridade dos dados armazenados.

Esta solução pode ser utilizada sob duas formas distintas: *Java Embeeded* e *Server Mode*. O acesso à base de dados em modo *Java Embeeded* é feito na mesma Java Virtual Machine (JVM) enquanto que o acesso em *Server Mode* é efetuado através da API Representational

[1] https://pt.wikipedia.org/wiki/LIFO
[2] https://pt.wikipedia.org/wiki/FIFO

State Transfer (REST). Torna-se então fácil de perceber que o segundo modo é mais flexível sendo que o mesma tende a possuir uma performance inferior[3].

A linguagem utilizada para aceder tipicamente à base de dados do Neo4j corresponde a Cypher. Corresponde a uma linguagem de query de grafos declarativa e bastante poderosa[4]. Se se utilizar a versão *embeeded* é também possível utilizar a API de Java diretamente; recorrendo a um paradigma *object-oriented* é possível, utilizando as entidades `Node`, `Relationship` e `Path`, criar algoritmos que atravessem o grafo altamente customizáveis. Na listagem 2.3 é apresentado um exemplo da utilização da API de Java, em que `graphdb` corresponde à instância do grafo.

```
1 ResourceIterable<Node> resulSet = graphdb.findNodesByLabelAndProperty (
    DynamicLabel.label(label), LABEL_NODE_VALUE, value);
2 ResourceIterator<Node> it = resulSet.iterator();
3 while (it.hasNext()) {
4     Node n = it.next();
5     System.out.println("Node : "+ n . getLabels().iterator().next() + " , Value
      : " + n.getProperty(LABEL_NODE_VALUE) ) ;
6 }
```

Listagem 2.3: Neo4j Core-Java-API Usage

2.2.3 HyperGraphDB

O HyperGraphDB, apresentado em [16], corresponde a uma base de dados de hipergrafos. De fato, estende este conceito na medida em que possibilita que arestas se conetem a outras arestas. Devido à complexidade dos modelos de dados que suporta, corresponde à escolha ideal para a resolução de problemas relacionados com inteligência artificial, bio-informática e processamento de linguagem natural.

A unidade básica de representação de informação é chamada de *atom*, átomo. No âmbito do hipergrafo, um átomo pode representar tanto um nó como uma aresta. Cada átomo tem associado a si um tuplo de outros átomos, designado de *target set*. Quando este tuplo tem uma dimensão igual a zero, o átomo é designado por nó (*node*). Caso contrário, é designado de aresta (*link*); ou seja, o conjunto nós abrangidos por uma aresta é especificado no *target set* da mesma. O conjunto de incidência de um nó (n_1) é assim composto pelo conjunto de átomos que contêm n_1 no seu *target set*.

Relativamente à informação armazenada num átomo, esta corresponde a um *strong value*. Quer isto dizer que o valor que está armazenado no átomo corresponde a uma entidade própria, que não está dependente do seu recipiente. Átomo e valor constituem assim, como referido em [16], dois aspetos ortogonais da informação armazenada numa instância de `HyperGraphDB`, correspondendo um átomo à entidades semântica que forma a estrutura da base de dados e os seus valores um tipo de dados que pode ou não ser estruturado.

Neste contexto surge a necessidade da utilização de um sistema encarregue de gerir estes mesmos dados, correspondendo ele próprio a uma estrutura de um hipergrafo. O HyperGraphDB é formado por duas camadas: uma de armazenamento primitivo e outra que modela a base de dados. A primeira é formada por dois *arrays*: um de identificadores e outro de dados não tratados:

[3] http://asakta.blogspot.pt/2014/04/experiences-using-graph-database-neo4j.html
[4] http://neo4j.com/docs/stable/cypher-introduction.htm

$LinkStore : ID \rightarrow List < ID >$
$DataStore : ID \rightarrow List < byte >$
A segunda camada é a responsável pela estrutura do hipergrafo, sendo formalizada da seguinte forma:
$AtomID \rightarrow [TypeID, ValueID, TargetID, ..., TargetID]$
$TypeID \rightarrow AtomID$
$TargetID \rightarrow AtomID$
$ValueID \rightarrow List < ID > | List < byte >$
Um átomo está associado a um tipo de dados, com o respetivo valor e os respetivos átomos que constituem o seu target set.

2.3 Controlo de Acesso

Nesta secção será introduzido o conceito de `Controlo de Acesso` bem como o seu propósito. A subseção 2.3.1 apresenta alguns dos modelos de controlo de acesso existentes. Em 2.3.3 é exposto brevemente o tipo de mecanismos usados para implementar ferramentas de controlo de acesso a baixo nível e exemplos dessas mesma ferramentas.

Por definição, controlo de acesso corresponde a uma técnica de segurança que permite regular o acesso de sujeitos a recursos que se pretendem proteger. A comunicação nestes casos é efetuada através de diálogos *request-response*, sendo que a mediação da mesma é realizada sob a forma do esquema apresentado na figura 2.3.

Figura 2.3: Controlo de Acesso: Modelo de Interação

Estes sistemas são normalmente responsáveis pelos processos de identificação e autenticação, autorização e auditoria. Funcionam como *middleware*, intercetando e analisando todas as tentativas de acesso aos recursos que se pretendem proteger. Visto que o âmbito desta dissertação está restringido à permissão de acesso a um recurso: autorização, será sobre ele que incidirá esta seção.

Como é dito em [24], estes sistemas têm normalmente que assegurar o `sigilo` de informação, ou seja, evitar que utilizadores não autorizados tenham acesso aos recursos; a `integridade` dos dados, ou seja, evitar que utilizadores não autorizados consigam modificar a informação armazenada e a `disponibilidade` de acesso aos recursos, ou seja, ao mesmo tempo que deve ser impedido o acesso não autorizado, os utilizadores autorizados devem tê-lo sempre que requisitarem.

Também de acordo com [24], o processo de desenvolvimento destes sistemas é realizado em várias fases baseando-se nos conceitos de `Políticas de Segurança`, onde as políticas de acesso são definidas, descrevendo que entidades podem aceder a que recursos e a quais não podem aceder; `Modelo de Segurança`, onde as políticas supramencionadas são formalmente definidas e posteriormente desenvolvidas. Este passo permite a prova das propriedades de

segurança fornecidas pelo sistema que se está a implementar. Por fim, o conceito de `Mecanismo de Segurança` está associado à definição a baixo-nível das funções que se encarregarão de implementar as políticas de acesso definidas anteriormente.

2.3.1 Modelos de Controlo de Acesso

Os modelos de controlo de acesso mais comuns são Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role-Based Access Control (RBAC) e Attribute-Based Access Control (ABAC). Serão apresentados nas subseções subsequentes.

Além destes, existem outros modelos, como por exemplo:

- Context-Based Access Control (CBAC), que lida com o controlo de acesso dos recursos partilhados recorrendo a um contexto flexível e dinâmico[25];
- Location-Based Access Control (LBAC), que utiliza a localização como um parâmetro para conceder ou negar acesso a um recurso[26];
- Intent-Based Access Control (IBAC), que no âmbito do processo de validação de um acesso a um recurso procura responder à pergunta *porquê?* o acesso está a ser requerido em vez à pergunta *quem?* está a pretender aceder ao recurso[27].

Controlo de acesso Discricionário

Neste modelo, a autorização de acesso a um determinado recurso protegido é baseada na identidade do utilizador que requereu o acesso[24] através da definição das operações (leitura, escrita, execução) que podem ser realizadas sobre cada recurso por cada utilizador. A origem da palavra discricionário no nome do modelo advém da possibilidade de certos utilizadores, mediante a autorização devida, poderem passar os seus privilégios sobre determinados recursos a outros utilizadores.

A definição das políticas de acesso a cada um dos recursos por parte de cada utilizador foi inicialmente proposta recorrendo a uma `matriz de controlo de acesso`, que é apresentada na tabela 2.4.

User	File X	File Y	File W	File Z	Programm P
A	read, write		read		execute, read
B	read		read,write		execute
C		read	read	read	

Tabela 2.4: Exemplo de matriz de controlo de acesso

Apesar desta solução, concetualmente estar correta, não é eficiente em termos de utilização de memória[24]. Por norma os utilizadores têm direitos de acesso a um conjunto restrito de recursos, fazendo com que a maioria das células apareçam vazias. Foram apresentadas três possíveis soluções apresentadas para este problema, onde eram armazenadas apenas as células ocupadas da matriz de controlo de acesso. A primeira consiste na criação de uma `tabela de autorização`, onde as células transportadas para uma tabela com três campos: sujeito, operação autorizada e recurso. Alternativamente podem ser utilizadas `listas de controlo de acesso`, onde a cada recurso está associada uma lista com as operações autorizadas por

utilizador. A última solução apresentada corresponde à criação de `listas de capacidade`, onde a cada utilizador está associada uma lista com as operações autorizadas por recurso.

Controlo de Acesso Obrigatório

Neste tipo de controlo de acesso, as políticas são estaticamente implementadas por um administrador e aplicadas por um mecanismo de controlo de acesso central[24]. A forma mais comum de definição das políticas de acesso corresponde à organização por nível de segurança. Aqui os sujeitos e os recursos são inseridos em níveis de segurança, podendo os sujeitos posteriormente aceder a recursos que sejam do mesmo nível de segurança que o seu ou inferior. Exemplos de níveis de segurança correspondem a Secreto, Confidencial, Reservado e Não Classificado.

Outra alternativa passa por atribuir os sujeitos e recursos a áreas de competências (Administração, Gestão, Pesquisa), podendo os recursos, por exemplo, estar inseridos em mais do que uma área. Posteriormente, para determinar se um sujeito pode aceder a um determinado recurso, é verificado se a área de competência do sujeito se interseta com a do recursos alvo.

Neste modelo, ao contrário do anterior, os utilizadores não podem delegar os seus privilégios a terceiros.

Controlo de Acesso baseado em Papéis

A necessidade deste modelo surgiu com a necessidade de especificar e fazer cumprir políticas de segurança de acordo com o esquema organizacional de uma empresa[24]. Nestes cenários, é mais importante saber-se a posição que um determinado utilizador exerce numa empresa do que propriamente saber quem o utilizador é. Neste âmbito, quando um utilizador requer acesso a um determinado recurso, fá-lo em nome do cargo que está a desempenhar no momento do acesso. Este modelo torna-se mais flexível do que os apresentados anteriormente na medida em que consegue conjugar a existência de autorizações específicas com a estrutura organizacional em que os utilizadores estão inseridos.

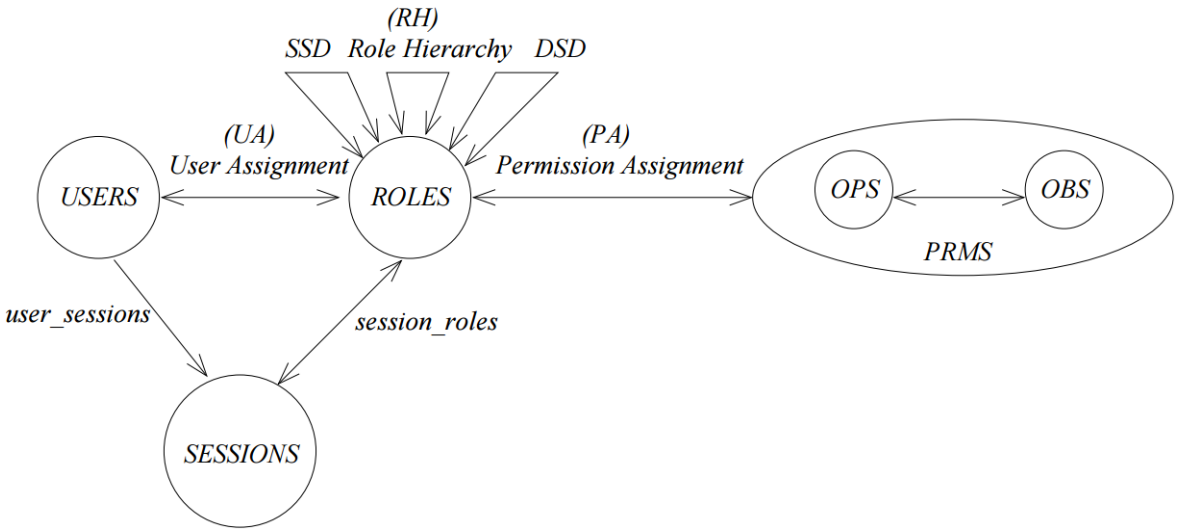

Figura 2.4: Modelo RBAC[28]

Tal como apresentado na figura 2.4, neste modelo, os utilizadores (USERS) estão associados a papéis (ROLES), sendo que as mesmas estão associadas a permissões (PRMS) que contêm o conjunto de operações passíveis de serem executadas (OPS) sobre o conjunto de informação protegida (OBS). Adicionalmente, como se pode depreender desta figura, este modelo inclui um conjunto de sessões (SESSIONS), onde em diferentes sessões os utilizadores podem estar associados a *roles* distintas[29].

Controlo de Acesso baseado em Atributos

Este modelo lógico de controlo de acesso recorre a atributos das entidades (sujeito e recurso), operações a realizar no acesso que se pretende avaliar e atributos de ambiente[30][31]. Estes atributos são comparados com as políticas, regras e relações entre os sujeitos e recursos definidas previamente para determinar a autorização/não autorização da operação em questão. Estas regras e políticas definem resultados booleanos complexos capazes de avaliar múltiplas combinações de atributos, permitindo um controlo de acesso mais preciso.

Este modo flexível uma vez que torna possível a criação de regras de acesso sem ter que especificar relações entre cada par (sujeito, recurso). É possível por exemplo a um funcionário de um determinado departamento de uma empresa ter acesso aos recursos desse mesmo departamento, desde que possua um atributo que o identifique. Além disto, caso caso se verifique alguma alteração, como por exemplo uma mudança de departamento/recrutamento de um funcionário, é possível redefinir as permissões de acesso ao mesmo necessitando apenas de alterar o valor do atributo em questão e sem ter que redefinir as regras de acesso ou relações entre o sujeito e o recurso.

Este modelo torna-se assim mais dinâmico e faz com que a manutenção do controlo de acesso a longo termo seja mais fácil.

2.3.2 Granularidade no Controlo de Acesso

Por definição, granularidade corresponde ao grau em que um sistema pode ser subdividido[5]. Tal como se pode inferir pelo nome de cada uma destas estratégias, um mecanismo de controlo de acesso de alto nível (`coarse-grained`) tende a trabalhar sobre items de maior granularidade, enquanto um mecanismo de baixo nível (`fine-grained`) tende a operar sobre subconjuntos desses mesmos items[6]. Transpondo este conceito para o mundo das bases de dados relacionais, pode-se associar controlo de acesso a alto nível com a permissão de acesso a tabelas, vistas ou tuplos enquanto que o controlo de acesso a baixo nível está tipicamente associado ao nível do atributo (coluna) ou ainda ao nível do valor de cada célula.

Relacionando a granularidade de um mecanismo de controlo de acesso com os modelos apresentados anteriormente (DAC, MAC, RBAC e ABAC), pode-se perceber que os três primeiros tendem a ser mecanismos de controlo de acesso de alto nível e o último de baixo nível. Tanto nas listas de controlo de acesso (DAC), como na definição dos papéis que um utilizador pode desempenhar (RBAC), as políticas de acesso tendem a tomar valores booleanos em que um utilizador pode, ou não, aceder a um determinado objeto.

Recorrendo a um exemplo[7]: para um cenário em que se pretende estabelecer de um determinado funcionário pode ou não abrir uma porta de um departamento, com o modelo

[5]https://en.wikipedia.org/wiki/Granularity
[6]http://www.webfarmr.eu/2011/05/coarse-grained-vs-fine-grained-access-control-part-i/
[7]http://www.webfarmr.eu/2011/05/coarse-grained-vs-fine-grained-access-control-part-i/

RBAC é possível definir uma *role* em que um funcionário, atuando em nome dessa mesma *role*, pode ou não aceder ao recurso protegido (porta do departamento). No entanto, caso se queira aplicar uma restrição que defina o período de tempo em que essa mesma *role* seja válida, ou então a operação em particular que o funcionário possa realizar, como por exemplo poder apenas abrir ou fechar uma porta, a solução não é tão flexível quanto o desejado. Para solucionar cada uma destas restrições seria necessário criar uma *role* diferente.

Recorrendo ao modelo ABAC, todas estas restrições podem ser acomodadas numa mesma política de privacidade em que são utilizados os atributos do sujeito, recurso, operação e de contexto, podendo assim ser efetuado um controlo de acesso especificando em concreto as operações (com as eventuais restrições) que podem ser efetuadas por um sujeito sobre um objeto.

2.3.3 Mecanismos de Controlo de Acesso de Baixo Nível

Existem várias técnicas que podem ser utilizadas para garantir o controlo de acesso a baixo nível (*fine-grained access controlol*- FGAC). Nesta subseção serão apresentadas as mais relevantes [32].

Reescrita de *Queries*

Apelida-se de reescrita de *queries* ao mecanismo que é responsável por decompor cada declaração de cada acesso a uma base de dados e analisar se a execução da mesma revela tuplos de informação definida nas políticas de acesso como sensível. Esta análise não é feita ao nível da informação efetivamente retornada, mas sim ao nível sintático da *query*, onde é verificado se alguma restrição que foi definida pelo administrador do sistema é violada.

Por exemplo, se tiver sido definida como informação sensível, no contexto da relação Employees, apresentada na tabela 2.2, todos os tuplos cujo valor do atributo $EmployeeID < 50$; se um utilizador inquirir a base de dados por todos os tuplos da tabela Employees, as soluções que implementam esta técnica serão responsáveis por, num primeiro momento verificar que uma política de privacidade será violada e posteriormente adicionar os predicado à declaração a ser executada (tipicamente na `cláusula where`) para que os possíveis tuplos que correspondem a informação sensível sejam filtrados. De igual forma, é também possível filtrar as colunas retornadas por uma query se as mesmas tiverem sido definidas como informação sensível.

Vistas e Vistas Parametrizadas

É possível divulgar apenas informação autorizada se para tal forem criadas vistas que filtrem a informação autorizada. Neste mecanismo, as queries são executadas sobre as vistas definidas em vez das tabelas originais. Esta técnica pode ser transparente ao utilizador comum, se utilizada em conjunto com a descrita anteriormente. Um utilizador pode inquirir uma tabela e o mecanismo, se detetar que a a mesma contém informação sensível, é responsável por alterar a query por forma a que ela inquira a vista que contém apenas informação autorizada.

Por vezes é necessário filtrar a informação sensível de acordo com os atributos do utilizador que está a efetuar o acesso à base de dados. Neste contexto surge o conceito de vistas parametrizadas. Por exemplo, ao ser definida uma política que permita a cada empregado consultar apenas os seus dados pessoais, quando um empregado inquire a tabela Employees este me-

canismo fica encarregue de aceder à informação autorizada através da vista parametrizada definida pela declaração apresentada na listagem 2.4.

```
1 create view MyPersonalInformation as select * from Employees where EmployeeID =
      \$user-id;
```

Listagem 2.4: Exemplo de declaração de criação de vista parametrizada

Valores Nulos

Além das 2 estratégias apresentadas previamente é possível também mascarar o conteúdo de informação sensível recorrendo a valores *null*. Apesar desta estratégia ser fácil de implementar é fácil perceber que através do fato de o resultado retornado corresponder a *null*, um utilizador pode facilmente perceber que se trata de informação sensível. Por si só, esta perceção pode, em muitos casos, constituir uma violação das políticas de privacidade estabelecidas.

2.3.4 Trabalho Relacionado

Nesta subseção serão apresentadas algumas soluções que efetuam controlo de acesso em modelos relacionais recorrendo aos mecanismos apresentados na subseção anterior.

Halder, 2010

Quando uma query é executada, é avaliada através de um mecanismo que verifica se a informação da base de dados pode ser revelada ou não. Neste âmbito divide-se a informação total armazenada em 2 grupos: sensível e não sensível. Quando é detetado que a query acede a informação sensível, a mesma não é divulgada, sendo retornada apenas a não sensível.

De acordo com a solução `Observation-based Fine-Grained Access Control` (OFGAC) apresentada em [33], a informação é disponibilizada a vários níveis de abstração. Neste contexto, os utilizadores, apesar de não conseguirem aceder ao valor exato da informação sensível, têm acesso a parte da mesma. A abstração é implementada sob a forma de funções, denominadas α, que são encarregues de transformar os valores que estão presentes na base de dados e devolver essas mesmas transformações em vez dos valores originais. Nos casos limite, estas funções retornam o valor de entrada (função identidade), no caso de se tratar de informação não sensível ou então retornam um valor/símbolo especial, no caso de se tratar de informação sensível com um grau elevado. Sob este ponto de vista, pode-se concluir que o modo tradicional FGAC corresponde a um caso particular do OFGAC.

Exemplificando, para o estado da base de dados apresentado na tabela 2.5:

Se as funções α_{salary} e $\alpha_{country}$ forem definidas da seguinte forma

$$\alpha_{salary}(\mu) = \begin{cases} \text{low} & \text{if } \mu \in [500, 1499] \\ \text{medium} & \text{if } \mu \in [1500, 2499] \\ \text{high} & \text{if } \mu \in [2500, 3499] \\ \text{very high} & \text{if } \mu \geq 3500 \end{cases}$$

$$\alpha_{country}(\mu) = nil$$

EnployeeID	Name	Country	Salary
1	Nancy	USA(N)	2954.55(N)
2	Andrew	USA(N)	2254.49
3	Janet	USA	3119.15(N)
4	Margaret	USA	1861.0(N)
5	Steven	UK(N)	1744.21

Tabela 2.5: Exemplo da Tabela Employees

EnployeeID	Name	Country	Salary
1	Nancy	nil	high
2	Andrew	nil	2254.49
3	Janet	USA	high
4	Margaret	USA	medium
5	Steven	nil	1744.21

Tabela 2.6: Tabela Employees abstrata

Quando o utilizador inquirir a tabela Employees através da query `select * from Employees;` os valores retornados serão os apresentados de seguida:

Através da função $\alpha_{country}$ percebe-se que o nível de abstração do atributo *country* é o mais elevado possível.

Neha Sehta, 2012

A linguagem SQL implementa mecanismos de controlo de acesso de alto nível, no sentido em que permite apenas controlar o acesso ao nível da base de dados, tabelas, vistas; sendo o nível de colunas (atributos) o nível mais baixo que *consegue ir*.

Esta solução[34] está assente no modelo de autorização do `System R` [35], em que o criador de um objeto (tabela ou vista) recebe autorização para executar todas as operações sobre o mesmo, podendo também autorizar o acesso através da instrução SQL GRANT. Implementa um modelo RBAC ao nível da base de dados, utilizando o mecanismo de reescrita de queries dinâmica. Funciona sobre o modelo *closed policy*, que suporta apenas políticas de autorização positivas. As queries são modificadas ao serem adicionadas regras às suas `cláusulas where` de acordo com as políticas de privacidade estabelecidas para as *roles* a que o utilizador está associado [34]. Para definir estas políticas de privacidade foram criadas tabelas adicionais que armazenam as restrições relativas às múltiplas *roles*. As restrições de cada política são armazenadas sob a `Forma Normal Conjuntiva`, em que a restrição é definida como uma conjunção de cláusulas, em que uma cláusula corresponde a uma disjunção de literais [8]. A arquitetura da solução desta ferramenta é apresentada na figura 2.5.

O módulo de `Autenticação e Autorização` é responsável por autenticar o utilizador e extrair da base de dados `User Info` as *roles* a que o mesmo está associado. A interface

[8] https://en.wikipedia.org/wiki/Conjunctive_normal_form

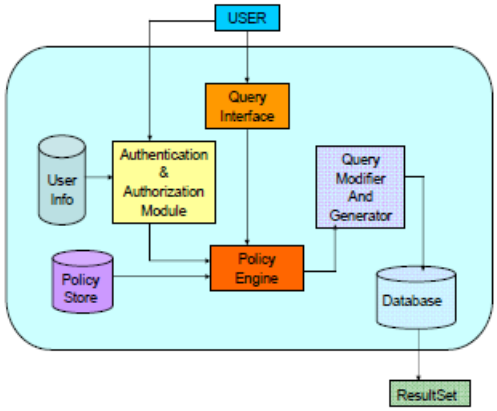

Figura 2.5: Visão Geral do Sistema[34]

de utilização está associada ao módulo `Query Interface`. Quando é efetuada uma query, o módulo `Policy Engine` é responsável por, de acordo com a *role* do utilizador e com as políticas extraídas do `Policy Store`, em conjunto com o módulo `Query Modifier and Generator` alterar a query de entrada e posteriormente executá-la sobre a base de dados.

São apresentados de seguidas 3 tabelas que, em conjunto, representam a forma como as políticas de privacidade são armazenadas.

PolicyID	ConditionID
P1	C1
P1	C2
P2	C1
P2	C3

Tabela 2.7: Tabela de Políticas

ConditionID	DescriptionID
C1	D1
C1	D2
C2	D3
C3	D4

Tabela 2.8: Tabela de Condições

De acordo com a informação da tabela 2.7 depreende-se que $P1 = C1 \wedge C2$. Por sua vez, recorrendo à informação armazenada em 2.8, $C1 = D1 \vee D2$ e $C2 = D3$, recorrendo posteriormente à tabela 2.9, pode-se concluir que a política de privacidade P1 é definida pela restrição

$$P1 = (Employees.Salary \leq 2000 \vee Employees.Salary \geq 4000) \wedge Employees.Country = USA$$

DescriptionID	OP1	RelOP	OP2
D1	Employees.Salary	$\leq$	2000
D2	Employees.Salary	$\geq$	4000
D3	Employees.Country	=	USA
D4	Employees.EmployeeID	$\leq$	10

Tabela 2.9: Tabela de Descrição

Ji-Won Byun

Tipicamente as políticas de privacidade estabelecidas baseiam-se no objetivo/propósito com que um determinado recurso será acedido, o tempo pelo qual estará alocado, as condições em que pode ser acedido e, por fim, as ações que devem ser realizadas após ser libertado. Purpose Based Access Control[36] implementa controlo de acesso recorrendo à noção de `propósito`, recorrendo a um modelo RBAC. Através desta propriedade consegue-se perceber *diretamente* a forma como a informação deverá ser acedida.

Normalmente, em ambientes empresariais, os propósitos, tal como as *roles*, são associados em relações hierárquicas, de forma a simplificar a gestão. Neste caso são utilizadas árvores invertidas em que a raiz da mesma representa a *role* mais geral e as permissões associadas a ela são herdadas pelos ramos originários a partir desse ponto.

Concetualmente, o acesso a um determinado recurso será permitido se o conjunto de propósitos definidos pelas políticas de privacidade estabelecidas contiver o propósito (ou um descendente seu, no supramencionado modelo hierárquico) com que o acesso está a ser efetuado.

Esta solução implementa o conceito de *intended purpose* para definir o conjunto de propósitos permitidos (*allowed intended purposes* - AIP) e os proibidos (*prohibited intended purposes* - PIP). Em caso de conflito, recorrendo ao princípio `denial-takes-precedence` onde o conjunto PIP se sobrepõe ao AIP.

Nesta solução, a estratégia utilizada para determinar o propósito acesso, está assente na premissa de que os utilizadores o devem explicitar a cada tentativa de acesso. Para o permitir, a linguagem sql foi estendida de forma a possibilitar que se adicionasse uma cláusula para especificar o propósito de acesso; por exemplo a query *SELECT CustomerID, Name FROM Customers;* transformar-se-ia em *SELECT CustomerID, Name FROM Customers FOR Billing*. Para facilitar a gestão da lista de propósitos associados a cada utilizador, os mesmos são associados às *roles* que ele desempenha. Este método ganha particular importância na medida em que grande parte dos sistemas definidos hoje em dia já estão a utilizar modelos RBAC para assegurarem controlo de acesso.

Além da noção de propósito, associados a ela estão também associados *role attributes*. É de seguida apresentado um exemplo desta organização.

Além dos *role attributes*, são também utilizados atributos de sistema para especificar eventuais estados do sistema em que as autorizações devem ser válidas. Recorrendo a um exemplo mencionado em [36], no qual se pretende dar permissão aos utilizadores da equipa de *E-Marketing* a aceder a informação com o propósito de "Service-Update", pode-se atribuir esse valor ao *role attribute* **ServiceType**. Posteriormente, assumindo que *timeofday* corresponde a um atributo de sistema define-se uma política de acesso ($ServiceType = "UpdateInfo" \wedge timeofday \geq 9 \wedge timeofday \leq 17$). Assim, apenas os utilizadores que estão associados à *role*

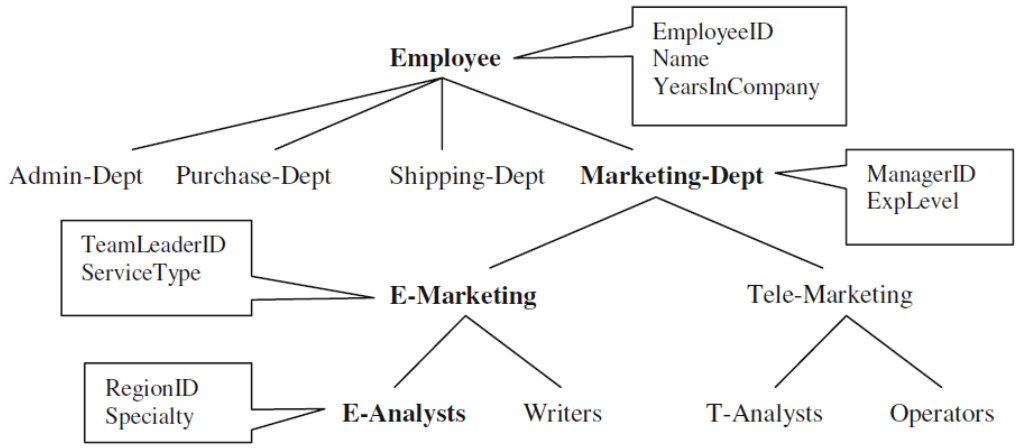

Figura 2.6: Árvore Hierárquica de Propósito[36]

E-Marketing (ou respetivos descendentes) podem aceder àquela informação durante o horário laboral.

Por fim, para se poder construir o modelo de controlo de acesso baseado na noção de propósito, basta apenas definir a forma como a informação pode ser catalogada. Para tal, os autores recorreram a um sistema de etiquetas onde utilizaram os *Intended Purposes*, sob a forma $< AIP, PIP >$. No âmbito do modelo relacional, é possível definir propósitos desde o nível de Tabela ao nível da célula de informação que está presente numa relação. É apresentado agora um exemplo da utilização de *labels* ao nível das colunas na tabela 2.10 utilizando a notação A Admin, P Purchase, S Shipping e M Marketing.

Table_name	Column_name	IP
Order	Product	$< \{A, P, S\}, \emptyset >$
Oder	Credit_info	$< \{P\}, \{M\} >$
Order	Date	$< \{A, P, S\}, \{M\} >$
Order	Status	$< \{A, P\}, \emptyset >$

Tabela 2.10: Política de Privacidade ao nível das colunas

Rizvi, 2004

O trabalho apresentado em [37] tem como base o mecanismo de transformação de queries, descrito previamente neste capítulo. Este mecanismo é transparente ao utilizador. Os autores recorrem ao conceito apelidado de *authorization views* (AV) para especificar o controlo de acesso, onde definem a informação que pode ser disponibilizada. Neste caso, as *queries* submetidas pelos utilizadores são aceites se puderem ser respondidas usando apenas informação que esteja contida nestas *views*.

Os utilizadores ao escreverem as queries podem fazê-lo sobre as tabelas originais. Posteriormente a ferramenta é encarregue de transformar as queries recebidas e tentar descobrir se são ou não válidas. Como uma primeira aproximação, uma query Q é definida como válida se houver alguma query Q', que utilizando apenas as AV definidas disponíveis, devolva o mesmo

resultado que Q. Ou seja, caso se trate de uma query condicional ou incondicionalmente válida, tal como os autores lhes chamam. Através de alguns conjuntos de regras, são demonstradas algumas formas de poder inferir se uma query é condicional ou incondicionalmente válida.

Entende-se por query incondicionalmente válida uma query Q, se houver uma query Q', descrita anteriormente, que para todos os estados da base de dados devolva o mesmo resultado que Q. Por outro lado, se a diferença entre Q e Q' depender do contexto da base de dados, chama-se a query Q de condicionalmente válida. Apesar de considerarem válida um query condicionalmente válida, e permitirem a sua execução, os autores reconhecem que esta solução pode, em circunstância especiais divulgam informação protegida.

Ao nível do mecanismo de transformação de queries, como já se pode ter percebido, apenas no âmbito de tentar avaliar a validade da query a ser executada. De fato, é fácil utilizando queries reconstruídas, perder-se performance na execução da query. Por isto, se a query original tiver sido validada pela ferramenta, é essa mesma query que é posteriormente executada sobre a base de dados.

Wang, 2007

No trabalho proposto em [38], os autores propõem um modelo baseado em etiquetas e um mecanismo de avaliação de *queries* para solucionar o problema do controlo de acesso a baixo nível.

Idealmente, um algoritmo que implemente FGAC deve retornar informação consistentemente mediante as queries que vão sendo efetuadas (de forma a que este mecanismo não seja detetado pelo utilizador), ao mesmo tempo que não revela informação definida como sensível e retorna o máximo de informação não sensível possível. Os autores alegam que esta solução respeita as duas primeiras propriedades e, relativamente à terceira, reconhecendo que é difícil de alcançar na prática, consideram que retorna tanta informação quanto as soluções existentes.

Relativamente ao mecanismo de utilização de *labels*, este é implementado ao nível de cada célula. São definidos 2 dois tipos de variáveis para auxiliar no processo de avaliação de uma *query*: o `tipo-1` corresponde a um símbolo num qualquer alfabeto. O `tipo-2` é formado por um tuplo $< \alpha, d >$, onde α e d correspondem a ao nome e ao domínio da variável, respetivamente. Estas *labels* são aplicadas a cada célula de cada tabela de uma base de dados por um algoritmo que atribui a cada célula de informação sensível uma variável do tipo-1, e quando essa célula pertence a um par de chave primária/estrangeira, uma variável do tipo-2, com o mesmo valor nas duas relações. Ou seja, para as duas tabelas 2.11 e 2.12, após a aplicação da máscara à tabela, o resultado é apresentado em 2.13 e 2.14

EnployeeID	Name	Country	Salary
1(N)	Nancy	USA(N)	2954.55(N)
2(N)	Andrew	USA(N)	2254.49

Tabela 2.11: Exemplo de tabela Employees

Relativamente ao algoritmo de avaliação implementado de forma a assegurar as 2 primeiras propriedades supramencionadas, o mesmo atua sobre as *máscaras* das tabelas inquiridas,

OrderID	CustomerID	EmployeeID	OrderDate
10248	VINET	1(N)	1996-07-04 00:00:00.00
10249	TOMSP	2(N)	1996-07-05 00:00:00.00

Tabela 2.12: Exemplo de tabela Orders

EnployeeID	Name	Country	Salary
$< \alpha, d_1 >$	Nancy	ν_1	ν_3
$< \beta, d_1 >$	Andrew	ν_2	2254.49

Tabela 2.13: Exemplo de tabela Employees mascarada

OrderID	CustomerID	EmployeeID	OrderDate
10248	VINET	$< \alpha, d_1 >$	1996-07-04 00:00:00.00
10249	TOMSP	$< \beta, d_1 >$	1996-07-05 00:00:00.00

Tabela 2.14: Exemplo de tabela Orders mascarada

sendo definidos dois tipos de avaliação: quando uma query simples é executada, aplica-se o método de `low evaluation` onde os tuplos que contenham células cuja informação é considerada sensível são tratados de forma tradicional, ou seja, removidos do resultado da query. Por outro lado, quando ocorre o caso de existir uma query composta por um operador *MINUS Q' - Q*, ao avaliar a query Q aplica-se o método `high evaluation`, onde o resultado dela necessita também de conter os tuplos que tenham células que correspondam a informação sensível.

Dorothy E. Denning, 1986

A solução apresentada em [39] recorre a vistas para fazer cumprir as políticas de acesso estabelecidas. A razão que levou os autores a recorrerem a vistas prendeu-se com o fato de elas corresponderem ao maior nível de abstração que se pode ter numa base de dados e, por tal, permitirem a especificação e aplicação de restrições ao nível de contexto (de acesso) e conteúdo (informação efetivamente armazenada na base de dados).

Foram definidas três tipos de vistas no âmbito deste trabalho: `classification constraints` que são responsáveis por atribuir *labels*, ao nível das células, a toda a informação que é registada na base de dados, `aggregation constraints` que restringem o acesso a agregados de informação e `access views` que permitem o acesso à informação de cada relação, que está limitada ao nível de segurança de um utilizador.

As *labels* que são atribuídas à informação correspondem a *classes de acesso*. Estas classes representam a classificação da informação em termos de níveis de segurança. Posto isto, para um dado utilizador poder aceder a uma dada informação, necessita que o eu nível de segurança seja igual ou superior ao da classe associada à informação a que pretende aceder. As classes de acesso podem ser atribuídas ao nível dos atributos (`type-dependent`), onde todas as células

de um determinado atributo ficam associadas a uma classe de acesso, ou então ao nível dos tuplos e/ou células em particular (`value-dependent`), sendo a sua classificação dependente do valor efetivo da informação que é armazenada.

Apesar de os autores reconhecerem que em termos do problema de agregação, a solução poder ser parcialmente resolvida com um desenho correto da base de dados, tal como já foi mencionado, são utilizadas também vistas próprias para tratarem deste problema, apelidadas de *aggregation constraints*. Estas vistas são definidas pelo conjunto de atributos que pretendem *proteger*, o número de tuplos que podem ser retornados e uma lista de utilizadores que estarão associados a estas restrições. Uma vista deste tipo definida por

"aggregation constraint Orders-AGG
Orders.OrderID, Orders.CustomerID, Orders.EmployeeID
count = 10 restrict Smith, Jones, Young"

limita o número de tuplos compostos pelos atributos `Orders.OrderID, Orders.CustomerID, Orders.EmployeeID` que os utilizadores Smith, Jones e Young podem aceder a 10.

Virtual Private Database

Virtual Private Database[40] baseia-se em técnicas de rescrita de queries antes da sua execução, de acordo com as políticas estabelecidas. Cada política de autorização é definida em funções para cada tabela; retornando cláusulas where a serem adicionadas às queries submetidas pelos utilizadores. Esta abordagem fornece uma perceção da base de dados diferente para cada utilizador.

A título de exemplo, numa base de dados de um hospital em que os médicos apenas podem ter acesso aos seus próprios pacientes, a função seria escrita para avaliar este predicado e a respetiva política de autorização seria definida ao nível da tabela Pacientes.

Esta solução, além de poder ser implementada filtrando as linhas que são retornadas pode também ser implementada de forma a filtrar/alterar a informação ao nível das colunas: removendo as colunas das queries ou, em alternativa, apresentando valores NULL para esses mesmos atributos.

LeFevre, 2004

Os autores em [41] propuseram uma técnica para controlo de acesso em bases de dados *hippocratic*[42] baseada em técnicas de reescrita de queries. O processo de divulgação de informação baseia-se na premissa de que o indivíduo tem controle sobre quem pode ver seus dados protegidos e com que finalidade. Como tal, as políticas compreendem um conjunto de regras que descrevem para quem os dados podem ser divulgados bem como com que propósito. Neste contexto, as políticas podem ser aplicadas ao nível de tabelas (sendo definidas vistas que substituem informação protegida por NULLs) e ao nível das queries submetidas (sendo removida a informação protegida dos resultados obtidos), de acordo com as restrições de propósito-recetor definidas.

S-DRACA

O autor em [43] desenvolveu um arquitetura distribuída, segura e dinâmica que toma em conta as políticas definidas para gerar mecanismos de segurança que são utilizados pe-

las aplicações. Estes mecanismos de segurança são utilizados para proteção de informação sensível, sendo que não possibilita sequer as respetivas aplicações a executar operações não permitidas. Esta solução fornece uma interface que as aplicações devem usar. Como esta interface é gerada e implementada em *runtime*, pode ser ajustada para refletir as alterações das políticas de acesso.

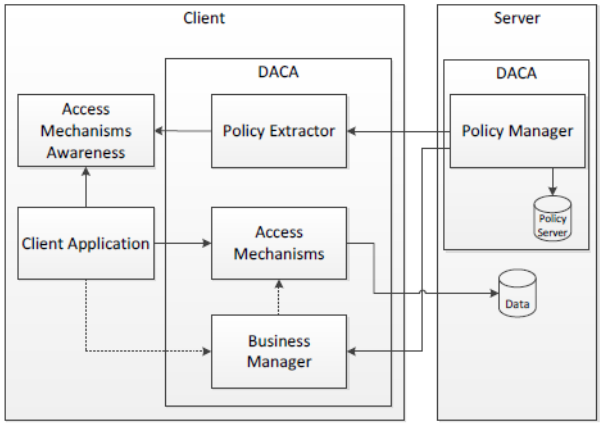

Figura 2.7: Arquitetura S-DRACA[43]

De fato, esta solução possibilita a definição de sequências válidas e inválidas de queries que os utilizadores podem ou não executar, correspondendo à culminação de diversos estudos efetuados previamente, como por exemplo [44][45][46]. A arquitetura desta solução é apresentada na figura 2.7. As políticas são armazenadas no `Policy Server`. Se o `Policy Manager` detetar que estas são alteradas notifica os clientes conetados. O `Policy Extractor` extrai as políticas do `Manager` e gera mecanismos através dos quais a aplicação cliente acede à base de dados.

Quando a aplicação cliente pretende aceder à base de dados, requer ao `Business Manager` que instancie esses mecanismos e utiliza-os. O componente `Business Manager` é também responsável por aplicar a execução das sequências válidas de queries. O componente `Authentication/Encryption` fornece mecanismos de autenticação e de cifra de informação.

Predicated Grants

Os autores em [47] propuseram uma generalização para o mecanismo de autorização de SQL. O modelo baseia-se em adicionar predicados aos comandos `Grant` e em estender o modelo de autorização SQL corrente para suportar autorização de baixo nível. A listagem 2.5 apresenta um exemplo simples.

```
1 grant select on Employees
2 where (employeeID=userId())
3 to public
```

Listagem 2.5: Comando grant[47]

Esta autorização especifica que cada empregado pode aceder apenas ao tuplo com a sua informação.

Este modelo também utiliza valores NULL para efetuar controlo de acesso ao nível das células, tal como é apresentado na listagem 2.6. Esta autorização especifica que o acesso ao atributo `addr` da tabela `Employees` é permitido caso o predicado seja satisfeito; caso contrário é retornado um valor NULL.

```
grant select on Employees(addr)
where (some predicate)
else nullify
to public
```

Listagem 2.6: Comando grant recorrendo a valores NULL[47]

A informação armazenada numa base de dados está associada a objetos do mundo real. O armazenamento destes objetos pode ser efetuado recorrendo a múltiplas tabelas. Neste contexto surge o conceito de autorização para grupos de tabelas relacionadas. Cada grupo de autorização deve possuir uma relação (tabela) raiz, tal como é apresentado na listagem 2.7.

```
create authorization sel_update_purchaseorder
with root order O as (
 update on order O,
 update, insert, delete on lineitem L
 where (L.order_id = order.order_id),
 select_purchaseorder O2
 where (O2.order_id = O.order_id));
```

Listagem 2.7: Criação de um grupo de autorização[47]

Posteriormente, este grupo de autorização pode ser atribuído a uma role, tal como é apresentado na listagem 2.8.

```
grant sel_update_purchaseorder
 where (purchaser_id in
 (select user_id from employee, manager
 where employee.deptid=manager.deptid
 and manager.mgrid = userId()))
to managerGrp;
```

Listagem 2.8: Atribuição de um grupo de autorização a uma Role[47]

Roichman, 2007

O trabalho desenvolvido em [48] tenta resolver o problema de ataques de injeção SQL[49]. Para superar essa lacuna de segurança, os autores propõem um mecanismo de controlo de acesso suportado por vistas parametrizadas. Para abordar a identificação dos utilizadores é apresentados um método baseado em parâmetros: a identificação de cada utilizador é conhecida (ou atribuída automaticamente) e usada para criar dinamicamente vistas parametrizadas que contêm a informação associada ao mesmo, evitando assim o acesso a dados não autorizados.

Esta abordagem impede assim que um utilizador possa aceder a informação da base de dados sem ser previamente identificado, visto que a informação a que ele acede é sempre obtida através de vistas parametrizadas. Posto isto, o processo de identificação torna-se um aspeto chave desta solução, mas insuficiente para abordar todos os aspetos de controlo de

acesso. Os próprios autores reconhecem que a metodologia proposta é restritiva porque não aborda todas as situações. Por exemplo, um usuário ainda pode manipular seus próprios dados ou informações.

λ_{DB}

Os autores em [50] apresentam a linguagem de programação λDB. Esta é utilizada para expressar e verificar políticas de controlo de acesso através de verificação de tipo estático. λDB introduz estruturas de programação (conhecidas com entidades) que definem tabelas da base de dados e as políticas associadas às mesmas. As expressões CRUD são posteriormente validadas contra as políticas estabelecidas (em tempo de compilação), tendo também em conta outras informações contextuais.

Cada política é composta por permissões que contêm as operações permitidas (escrita ou leitura), as listas de atributos e as condições booleanas em que as mesmas são permitidas. Na listagem 2.9 é apresentada a entidade `Person` com os atributos `userid`, `public`, `photo` e `secret`, sendo definida uma permissão para cada atributo. A título de exemplo o atributo `public` pode ser lido em todas as condições enquanto que `secret` apenas pode ser lido se o utilizador estiver autenticado no sistema e caso for o utilizador identificado naquele tuplo.

```
1 entity Person [ userid: string; public: string; photo: picture; secret: string]
2 read public where true;
3 read userid where Auth(uid);
4 read secret where Auth(uid) and uid = userid;
5 read photo where Auth(uid) and Friends(userid,uid);
6 write where Auth(userid);
```

Listagem 2.9: Definição de uma política λ_{DB}[50]

Java EE

Java EE [51] suporta a aplicação de políticas RBAC através de anotações[52], que são colocadas na definição de métodos para controlar quem tem permissão para os invocar. No exemplo apresentado na listagem 2.10 apenas os utilizadores com o papel `Administrator` podem invocar o método `setNewRate`, enquanto que o método `convertCurrency` pode ser invocado por todos os utilizadores.

```
1 import javax.annotation.security.*;
2 @RolesAllowed("RestrictedUsers")
3 public class Calculator {
4     @RolesAllowed("Administrator")
5     public void setNewRate(int rate) { ... }
6     @PermitAll
7     public long convertCurrency(long amount) { ... }
8 }
```

Listagem 2.10: Exemplo da utilização de anotações[53]

Esta solução não visa as expressões CRUD diretamente. No entanto, os objetos que são utilizados no âmbito dos respetivos métodos/classes podem conter expressões CRUD.

Apesar de estas políticas serem construídas em tempo de execução, os programadores estão conscientes das mesmas aquando do desenvolvimento das aplicações. Isto faz com que

os programadores não conseguem verificar de forma estática se a forma como a aplicação está desenhada respeita as políticas RBAC estabelecidas.

Test4Privacy

A solução proposta em [3] tem como principal objetivo ajudar os administradores de base de dados (ainda na fase de desenvolvimento) a detetar conjuntos de operações legais que permitam a um utilizador obter informação protegida. Mediante os resultados apresentados por esta ferramenta, o administrador pode alterar o desenho da base de dados ou redefinir os casos de utilização que possam violar as políticas de privacidade estabelecidas. As políticas de controlo de acesso são queries SQL, correspondendo a informação sensível ao resultado da execução das mesmas sobre a base de dados. Pode-se então perceber que esta solução é *content-dependent*. Esta solução correspondeu ao ponto de partida desta dissertação, na medida em que os objetivos de ambas são coincidentes: detetar situações nas quais o conjunto de operações autorizadas permite a um utilizador inferir informação sensível.

Adicionalmente, as políticas de controlo de acesso no âmbito da solução desenvolvida nesta dissertação são também definidas sob a forma de queries SQL. No entanto, tal como referido no capítulo 1, é permitida a definição de uma probabilidade associada às políticas de controlo de acesso definidas.

Os resultados em [3] são apresentados sob a forma de um relatório contendo o conjunto de queries que contribuiu para a quebra da(s) política(s) estabelecida(s), a(s) política(s) de privacidade violada(s), os tuplos de informação sensível que teriam sido revelados e a percentagem que essa informação sensível corresponderia.

O modelo desta solução é apresentado na figura 2.8. O módulo `requestor` fornece o conjunto de queries legais passíveis de serem realizadas e no final recebe um relatório da execução; o módulo `Test4Privacy`, forma os possíveis cenários de teste (com base nas queries fornecidas pelo `requestor`) e executa cada um sobre `Database` e posteriormente utilizando `Rules` (onde são armazenadas as queries que definem as políticas de controlo de acesso) avalia se as mesmas violam as políticas estabelecidas.

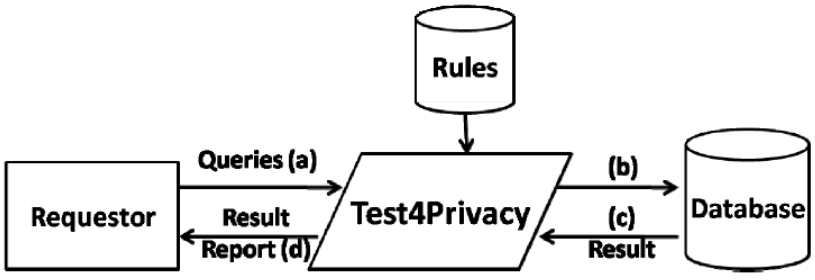

Figura 2.8: Modelo Test4Privacy[3]

O processo de teste é composto por duas fases; a primeira em que são formados os casos de teste (designada de `query aggregation`) e na segunda onde são avaliados os resultados (designada de `query evaluation`).

A primeira fase é realizada através do algoritmo `Next Aggregated Result` que é responsável por agregar as queries de teste de forma a descobrir possíveis oportunidades de

violar as políticas de privacidade estabelecidas. Este algoritmo produz as queries de uma forma iterativa, agregando-as segundo as regras:

- q_i [MINUS — INTERSECTION — UNION] q_j,$if X_i \equiv X_j$;
- q_i NATURAL JOIN q_j,$if X_i \cap X_j \neq \emptyset$;
- q_i CARTESIAN PRODUCT q_j.

em que X_i corresponde ao *schema* dos atributos inquiridos por q_i.

Comparando novamente a solução apresentada em [3] com a realizada no âmbito desta dissertação pode-se perceber que relativamente aos dois métodos de inferência de informação, a Test4Privacy utiliza técnicas de reescrita de queries no momento da fase `query aggregation`, correspondendo o resultado da execução dessas queries à informação considerada passível de ser inferida. No âmbito da solução desenvolvida nesta dissertação, foram utilizados grafos para armazenar a informação obtida a partir de cada query, representando posteriormente o atravessamento do grafo (mediante algumas condições) o processo de inferência passível de ser realizado por um utilizador.

Na segunda fase é aplicada a teoria de conjuntos para avaliar as queries geradas em *query aggregation* (q_{aggr}). Os *result sets* delas são comparados aos das queries que definem as políticas de controlo de acesso (q_{ill}). Ao comparar os *result Sets* pode surgir o caso de possuírem *schemas* iguais ou diferentes. No caso de o *schema* de X_{aggr} ser igual ao schema X_{ill} podem ocorrer as seguintes situações em que existe a possibilidade de as políticas de de controlo de acesso serem violadas:

- SAME_BLOCK, if $RS_{aggr} \subseteq RS_{ill}$
- SAME_COUNTERACTION, if $RS_{aggr} \supseteq RS_{ill}$
- SAME_INTERSECTION. if $RS_{aggr} \cap RS_{ill} \neq \emptyset$

Caso exista uma projeção π_X que contenha atributos de q_{aggr} e q_{ill}, podem ocorrer as seguintes situações em que existe a possibilidade de as políticas de controlo de acesso serem violadas:

- DIFF_WARNING, if $\pi_X(RS_{aggr}) \subseteq \pi_X(RS_{ill})$
- DIFF_COUNTERACTION, if $\pi_X(RS_{aggr}) \supseteq \pi_X(RS_{ill})$
- DIFF_INTERSECTION. if $\pi_X(RS_{aggr}) \cap \pi_X(RS_{ill}) \neq \emptyset$

Ao contrário desta solução, onde o paradigma passsa por comparar os *result sets* de q_{ill} com os resultados obtidos na fase *query aggregation*, pretende-se que o trabalho desenvolvido nesta dissertação avalie, para cada tuplo composto pelos atributos presentes na cláusula select de q_{ill}, se os valores inferidos dos atributos pertencentes à cláusula where de q_{ill} são válidos/ inválidos no âmbito da mesma.

Tal como foi mencionado, o objetivo destas ferramentas é detetar possíveis casos de uso onde possa ocorrer violação de políticas de controlo de acesso, caindo fora do âmbito a apresentação da solução para os casos supramencionados.

2.4 Java

A linguagem de programação Java é uma linguagem de propósito geral, concorrente, baseada em classes e orientada a objetos[4]. A sintaxe da mesma deriva em grande parte das linguagens C e C++.

Para viver num mundo de comércio eletrónico e distribuição, esta linguagem teve que possibilitar o desenvolvimento de aplicações seguras, robustas e de alto desempenho em plataformas heterogéneas e distribuídas, ou seja, teve que ser[54]:

1. simples, orientada a objetos e familiar;
2. robusta e segura
3. independente da arquitetura e Portátil;
4. elevado desempenho;
5. interpretada, *threaded* e dinâmica.

2.4.1 JDBC

Java Database Connectivity (JDBC) corresponde a uma API Java para execução de declarações SQL[55].

Foi desenvolvida pela `JavaSoft` com o intuito de ser[56]:

- uma API ao nível de SQL;
- simples;
- similar às APIs de bases de dados existentes.

Por o JDBC dever ser uma API ao nível de SQL, os autores referem-se ao fato de ser possível construir declarações SQL e incorporá-las dentro de métodos da API Java, tornando assim a transição entre o nível da base de dados e o nível da aplicação mais suave.

A figura 2.9 apresenta a arquitetura do JDBC tipo 4, que corresponde a uma solução 100% Java, fornecida pelo fabricante da respetiva base de dados (no âmbito desta dissertação foi utilizado o MySQL).

O JDBC driver é responsável por implementar o protocolo para a transferência das declarações SQL e os respetivos resultados entre o lado do cliente e a base de dados. A forma como estes drivers estão implementados é transparente às aplicações cliente que os utilizam. Isto permite que a mesma aplicação Java consiga comunicar com uma variedade de bases de dados diferentes, visto que os drivers são descarregados dinamicamente adequando-se à instância da base de dados relacional à qual se pretende aceder.

A título de exemplo, é apresentado na listagem 2.11 um trecho de código em que uma aplicação, recorrendo ao JDBC:

1. estabelece uma conexão com a base de dados;
2. envia uma declaração SQL;
3. processa os resultados obtidos.

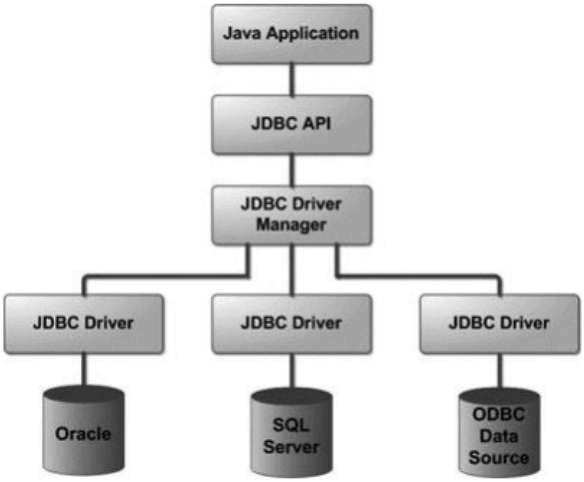

Figura 2.9: Arquitetura JDBC[57]

```
Connection con  = DriveManager.getConnection("jdbc:odbc:wombat","login","
    password");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c From Table1");
while(rs.next){
    int a = rs.getInt("a");
    String b = rs.getString("b");
    float c = rs.getFloat("c");
}
```

Listagem 2.11: Exemplo de utilização do JDBC[55]

Capítulo 3

Solução concetual

Tal como mencionado no capítulo 1, esta plataforma tem como principal objetivo detetar possíveis casos de uso autorizados em que informação sensível possa ser inferida. Com ela pretende-se apenas tratar o controlo de acesso ao nível das expressões de leitura (SELECT), e não de criação (CREATE), atualização (UPDATE) ou remoção (REMOVE).

Ao longo deste capítulo será apresentado e descrito a evolução das diferentes estratégias e abordagens utilizadas no âmbito da solução do problema proposto nesta dissertação.

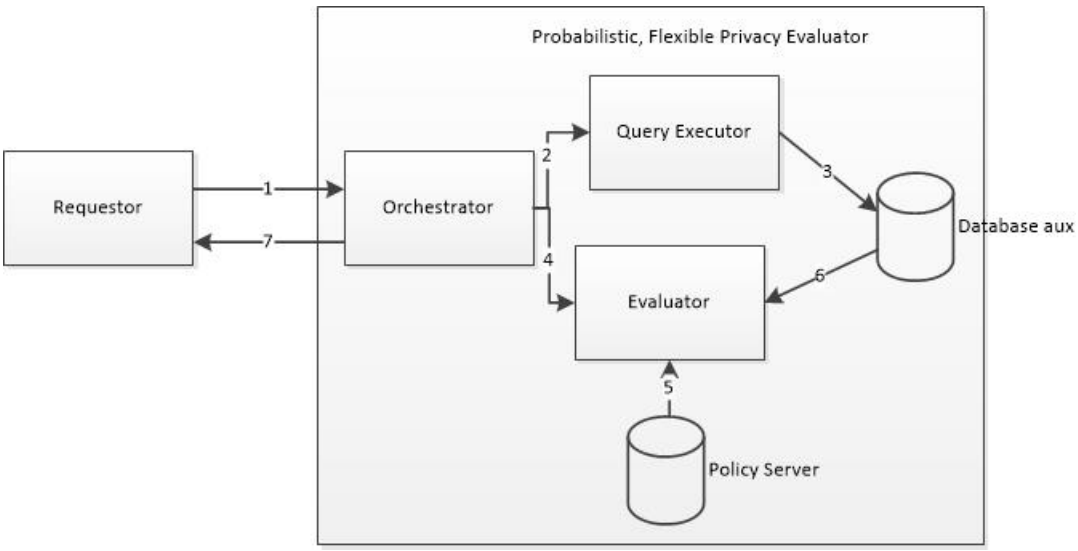

Figura 3.1: Avaliador de Privacidade Probabilístico e Flexível

A figura 3.1 apresenta o modelo da solução Avaliador de Privacidade Probabilístico e Flexível (*Probabilistic, Flexible Privacy Evaluator* PFPE). Tal como mencionado anteriormente, o processo de avaliação é realizado sobre políticas de controlo de acesso probabilísticas, sendo flexível tanto ao nível da granularidade como ao nível dos momentos em que pode ser efetuado.

O `Requestor` fornece o conjunto de queries autorizadas, que são posteriormente processadas por `Orchestrator`. É também através deste último que é retornado o resultado do processo de avaliação das queries submetidas. O `Query Executor` executa as queries que lhe são requeridas pelo `Orchestrator` e armazena os resultados das mesmas na base de dados

auxiliar `Database aux`. Por seu lado, o `Evaluator` avalia se com a informação obtida a partir das queries já executadas, que se encontra em `Database aux`, é possível violar as políticas de controlo de acesso estabelecidas no componente `PolicyServer`.

Caso durante a execução se verifique que as políticas de controlo de acesso são violadas no contexto do caso submetido pelo administrador, é produzido um relatório. Nele é explicitado **(1)** os tuplos sensíveis protegidos, aos quais o utilizador pode ter acesso através de técnicas de inferência associando-as a valores críticos de atributos sensíveis de predicado de q_{ill}; **(2)** o conjunto de queries utilizadas para violar a política de controlo de acesso estabelecida.

Como foi explicado no âmbito do capítulo 1, é aconselhável que a cada execução de uma nova query seja efetuado o processo de avaliação, apesar de não ser obrigatório. Esta característica torna então a solução mais flexível, ao permitir que um administrador possa ir redesenhando o caso de uso que pretende testar à medida que o vai construindo, consoante os resultados obtidos dos sucessivos processos de avaliação.

De acordo com a breve descrição desta solução apresentada no âmbito figura 3.1, pode-se também constatar que a mesma usa uma base de dados auxiliar. Nela, como mencionado, são armazenados os resultados das queries submetidas por `Requestor`. Tendo isto em conta, esta solução apenas trabalha com bases de dados estáticas, tanto em termos de esquema da base de dados como da informação efetivamente armazenada. A forma como esta informação é armazenada será abordada ao longo da seção 3.2.

O fato de se ter que lidar com uma base de dados dinâmica, implicaria o estudo e criação mecanismos que permitissem responder aos tipos de alterações supramencionadas. As alterações que fossem efetuadas ao nível do esquema da base de dados original teriam que ser repercutidas na base de dados auxiliar e as inconsistências geradas pela inserção, atualização ou remoção de informação à base de dados original tratadas. Considere-se o exemplo apresentado anteriormente na tabela 3.1. Supondo que são executadas, na ordem cronológica apresentada, as seguintes queries:

```
1 q1: select EmployeeID, Salary from Employees where Country = 'USA';
2 qup: update Employees set Salary = 1900 where EmployeeID = 1;
3 q2: select FirstName, Country from Employees where Salary > 2000;
```

Listagem 3.1: Definição de queries em cenário dinâmico

Na execução de q_1 é retornado o tuplo (EmployeeID,Salary) = (1, 2954.55). Por seu lado, após a execução de q_{up}, ao executar q_2, não é devolvido o tuplo (FirstName, Country) = ('Nancy', 'USA'), que corresponde ao tuplo associado a EmployeeID = 1. Ora, neste caso gera-se uma inconsistência que deveria ser devidamente suportada pela solução desenvolvida no âmbito desta dissertação. O foco desta dissertação não passou por resolver estes problemas mas sim efetivamente reconhecer e detetar convenientemente padrões de técnicas de inferência que pudessem levar à quebra das políticas de controlo de acesso. Optou-se então por trabalhar com uma instância estática da base de dados.

3.1 Definição de políticas de controlo de acesso probabilísticas

Nesta seção serão apresentadas primeiramente a forma como as políticas de controlo de acesso probabilísticas são definidas em 3.1.1. Posteriormente serão abordadas as estratégias

utilizadas com o objetivo de tornar esta solução mais flexível, sob a forma dos atributos utilizados no processo de avaliação em 3.1.2.

3.1.1 Definição de políticas de controlo de acesso probabilísticas

Este modelo funciona no modo *open policy*, onde apenas é permitido especificar políticas de autorização negativas, ou seja, definir a informação que não pode ser acedida pelos utilizadores. Recordando a motivação deste trabalho, apresentada no capítulo 1, no âmbito desta dissertação não existe quebra das políticas de controlo de acesso quando se acede a informação sensível mas sim quando se consegue associar a respetiva informação a valores de atributos sensíveis de predicado de q_{ill} válidos no âmbito da respetiva cláusula where. A definição dos vários componentes de informação sensível será apresentada de seguida.

As políticas de controlo de acesso mencionadas na seção 1.2, tal como referido anteriormente, podem ser descritas sob a forma de queries SQL. Serão designadas de q_{ill}, correspondendo a informação retornada por essas mesmas queries a informação considerada sensível protegida. Daqui para a frente serão designados de `atributos sensíveis` os atributos inquiridos na cláusula select de q_{ill}. O conjunto destes atributos foi designado `tuplo sensível`. Por outro lado, os atributos presentes na cláusula where de q_{ill} são designados de `atributos sensíveis de predicado`.

Os `tuplos sensíveis` podem ser divididos em dois conjuntos: `tuplos sensíveis protegidos` e `tuplos sensíveis não protegidos`. Os primeiros correspondem efetivamente aos tuplos retornados por q_{ill}, que são válidos no contexto da cláusula where da mesma. Os `tuplos sensíveis não protegidos` correspondem aos restantes tuplos presentes na base de dados mas que não são válidos no contexto da cláusula where de q_{ill} e, por conseguinte, não são retornados pela mesma.

A título de exemplo, a definição das políticas de controlo de acesso supramencionadas é apresentada na listagem 3.2. Adicionalmente, a cada uma está posteriormente associada uma probabilidade limite de acerto que varia entre 0 e 100%, que não é representada nesta listagem.

```
1 q_ill1: select EmployeeID, LastName, FirstName from Employees where Salary > 2500;
2 q_ill2: select CustomerID, CompanyName, ContactName from Customers join Orders on
    Customers.CustomerID = Orders.CustomerID join OrderDetails on Orders.OrderID
    = OrderDetails.OrderID where OrderDetails.UnitPrice > V1;
3 q_ill3: select CupplierID, CompanyName, ContactName from Suppliers join Products
    on Suppliers.SupplierID = Products.SupplierID where Products.Category = '
    category1';
```

Listagem 3.2: Exemplo da definição de políticas de controlo de acesso

Considere-se agora a porção de uma base de dados cujo estado é representado na tabela 3.1. Suponha-se também que sobre esta base de dados o administrador pretende aplicar uma política de controlo de acesso em que *não é possível saber que funcionários possuem um salário superior ou igual a 2500.00*, definida pela query q_{ill1}, apresentada na listagem 3.2. Convém novamente realçar que o objetivo desta solução não passa por impedir que um utilizador possa aceder aos tuplos sensíveis protegidos mas sim prevenir que um utilizador malicioso consiga perceber que um determinado tuplo sensível protegido está efetivamente associado a valores de atributos sensíveis de predicado válidos no contexto fornecido pela cláusula where de q_{ill}.

No âmbito desta política de controlo de acesso, o tuplo sensível é composto pelos atributos { EmployeeID, LastName, FirstName}. Os conjuntos de tuplos sensíveis protegidos

EmployeeID	LastName	FirstName	Country	Salary
1	Davolio	Nancy	USA	2954.55
2	Fuller	Andrew	USA	2254.49
3	Peacock	Margaret	USA	1861.08
4	Leverling	Janet	USA	3119.15
5	Buchanan	Steven	UK	1744.21
6	Suyama	Michael	UK	2004.07
7	King	Robert	UK	1991.55
8	Callahan	Laura	USA	2100.5
9	Dodsworth	Anne	UK	2333.33

Tabela 3.1: Exemplo da tabela Employees (2)

e não protegidos são apresentados nas tabelas `Tuplos Sensíveis` 3.2 da esquerda e direita, respetivamente.

EmployeeID	LastName	FirstName
1	Davolio	Nancy
4	Leverling	Janet

EmployeeID	LastName	FirstName
2	Fuller	Andrew
3	Peacock	Margaret
5	Buchanan	Steven
6	Suyama	Michael
7	King	Robert
8	Callahan	Laura
9	Dodsworth	Anne

Tabela 3.2: Tuplos Sensíveis

Recorrendo a novos exemplos concetuais, para o esquema da base de dados apresentada na figura A.1, apresentado no Anexo `A`, as políticas de acesso que concetualmente um administrador de uma base de dados pretende implementar (aplicáveis no âmbito desta solução), correspondem tipicamente a políticas que tentam evitar que um utilizador *malicioso* perceba

- quais os **empregados** (employees) que possuem um salário superior a um dado valor,
- quais os **clientes** (customers) que efetuaram compras superiores a um dado valor,
- quais os **clientes** (customers) efetuaram compras numa determinada data,
- quais os **fornecedores** (suppliers) que fornecem produtos pertencentes a uma determinada categoria.

De igual forma, um administrador de uma base de dados de um hospital, pretende prevenir tipicamente casos em que eventuais utilizadores *maliciosos* consigam perceber *quais os* ***pacientes*** *que têm uma determinada doença.*

Posto isto, pode-se perceber que estes empregados, clientes, fornecedores, pacientes correspondem a entidades: linhas de tabelas no âmbito da base de dados que se pretendem

proteger. Tendo isto em mente, e também que no modelo relacional é recorrendo a uma chave candidata de uma tabela que se torna possível identificar inequivocamente uma linha da mesma, foi definido como necessário que um tuplo sensível contenha esta chave.

Neste contexto, torna-se então recomendado que a base de dados recorra a chaves substitutas (surrogate keys) [1], cujos valores são independentes do contexto da informação armazenada. A utilização destas chaves pode ser vantajosa, por exemplo, quando o acesso às chaves naturais é considerado crítico, sendo assim desejável proteger essa mesma informação.

Relativamente ao esquema da base de dados, este deve ser normalizado. Este processo é aconselhável uma vez que com ele é possível obter uma redução da redundância de informação e o modelo desenvolvido apenas consegue detetar a associação entre atributos das mesmas tabelas quando as chaves estrangeiras estão devidamente declaradas.

Tal como já mencionado, além de não permitir a um utilizador conseguir associar os tuplos sensíveis protegidos aos respetivos valores de atributos sensíveis de predicado, em certos cenários é também importante que este não consiga executar um conjunto queries que o coloquem numa posição de ter uma probabilidade maior do que um determinado *limite*, definido pelo administrador, de acertar nesses tuplos. Posto isto, foi decidido que esta solução além de detetar quando um utilizador se encontra numa posição de conseguir identificar um determinado `tuplo sensível protegido`, também deveria permitir detetar quando existe uma probabilidade maior que um *limite* de acertar cegamente nesse mesmo tuplo; surgindo assim, no âmbito desta solução, o conceito de política de acesso probabilística.

Tal como mencionado no capítulo 1, estas políticas de controlo de acesso são definidas sob a forma de

<política de controlo de acesso> <probabilidade limite de acerto>

O cálculo desta probabilidade é dado pela expressão

$$\textbf{threshold} \leq \frac{\#\textbf{sensitive\_accessed\_tuples}}{\#\textbf{accessed\_tuples} - \#\textbf{non\_sensitive\_tuples}}$$

em que **threshold** $\in [0, 1]$ corresponde ao limite definido pelo administrador, #**accessed_tuples** corresponde ao número total de tuplos sensíveis a que o utilizador acedeu, #**non_sensitive_tuples** ao número total de tuplos sensíveis não protegidos que o utilizador já identificou como tal e #**sensitive_accessed_tuples** ao número de tuplos sensíveis protegidos a que o utilizador já teve acesso (apesar de não os ter identificado). Este **threshold** está associado a cada política de controlo de acesso, sendo aplicado a todos os `tuplos sensíveis protegidos` no âmbito da mesma.

Apresentando um exemplo em concreto, para o estado da base de dados apresentado na tabela 3.1, pretende-se implementar uma política de controlo de acesso definida pela query q_{ill1} e que o *threshold* definido é 0.5. Para o caso de uso definido pelas queries q_1 e q_2,

```
1 q1: select EmployeeID, FirstName, LastName from Employees;
2 q2: select Salary from Employees where EmployeeID > 4;
```

Listagem 3.3: Definição das queries autorizadas

[1] https://en.wikipedia.org/wiki/Surrogate_key

o utilizador tem acesso a todos os tuplos sensíveis disponíveis (através de q_1). Posteriormente ao inquirir a tabela com a query q_2, apesar de não ficar a saber que empregado em particular possui um salário superior a 2500, fica a saber que os empregados com o valor de EmployeeID superior a 4 correspondem todos a tuplos sensíveis não protegidos no âmbito de q_{ill1}. Perante este casso e recorrendo à terminologia utilizada na fórmula que calcula a probabilidade de um utilizador acertar num tuplo sensível protegido, #**accessed_tuples**=9, #**non_sensitive_tuples**=5 e #**sensitive_accessed_tuples**=2. Nesta situação uma violação da política de controlo de acesso ocorre, uma vez que existe uma probabilidade $100 * \frac{2}{9-5} = 50\%$ de acertar num tuplo sensível, que é igual ao *threshold* definido pelo administrador.

3.1.2 Flexibilidade na utilização de atributos no processo de avaliação

Esta solução tende a requerer um processamento exaustivo. A forma como se pretende dar alguma flexibilidade à mesma passou pela possibilidade de o administrador poder definir um conjunto de atributos que não devem ser utilizados no processo de avaliação. De fato, esta abordagem não consiste em literalmente ignorar estes atributos marcados pelo administrador, mas sim em obter o tuplo inteiro da respetiva tabela quando um destes atributos é inquirido pela query submetida pelo `Requestor`. O fato de obter então o valor de todos os atributos da tabela a que pertence o atributo marcado pelo administrador, contendo chave primária da mesma, facilita o processo de avaliação tal como se poderá constatar ao longo da seção `Processo de Avaliação` 3.3.

À medida que o administrador acrescenta mais atributos ao conjunto dos que não devem ser processados, a granularidade do controlo de acesso efetuado vai aumentando. De fato, num caso limite, o controlo de acesso é feito ao nível da linha (tuplo), em vez de ao nível da célula (valores dos atributos). Esta granularidade corresponde à granularidade das soluções que tipicamente utilizam vistas (parametrizadas) e/ou funcionalidades de reescrita de queries, tal como foi apresentado em 2.3.

Como será fácil de perceber, os únicos atributos que não podem ser adicionados ao conjunto dos que não devem ser processados correspondem às chaves primárias de cada tabela.

3.2 Armazenamento de Informação

Nesta seção será apresentado o paradigma utilizado no armazenamento de informação.

Para tal é necessário começar por refletir sobre a base deste trabalho, que consiste em tentar associar valores dos atributos sensíveis a valores de atributos sensíveis de predicado, no âmbito de uma política de controlo de acesso. No âmbito do mesmo (problema), esta associação pode não ser direta, mas sim através da utilização de diversas técnicas de inferência, que serão apresentadas ao longo da subseção 3.3. Como descrito na seção 2.2, sabe-se também que, tipicamente, problemas que envolvam relações entre várias entidades, podem ser representados recorrendo a teoria de grafos. Tendo então a tipologia deste problema em conta, o paradigma que foi escolhido para armazenar a informação obtida pelas queries executadas e posteriormente, efetuar o processo de avaliação, correspondeu à teoria de grafos. De fato, a solução final desenvolvida no âmbito desta dissertação corresponde a uma extensão deste paradigma. Como já mencionado, recorreu-se a hipergrafos, onde o princípio da conetividade é estendido a mais nós na medida em que uma aresta permite a conexão de mais do que dois nós.

Tendo em conta esta paradigma, atribuiu-se aos nós do grafo os valores dos atributos inquiridos, enquanto que as relações existentes entre os diferentes nós simbolizam a associação entre os mesmos. Caso ocorram múltiplas ocorrências de valores iguais no contexto de um atributo, recorre-se sempre ao mesmo nó. A identificação dos diferentes nomes dos atributos é efetuada sob a forma de

<table_name>.<attribute_name>

De fato, é necessário ter em conta a identificação dos atributos que correspondem a chaves estrangeiras. Se no âmbito de uma query for inquirido um destes atributos, ao registar os respetivos valores na base de dados auxiliar, deve-se registá-los com a identificação da sua tabela e nome originais.

Designe-se de `Exemplo 1`, definido na listagem 3.4, a um exemplo simples do tipo de associação que se pretende obter. Suponha-se também que se está a trabalhar com uma base de dados composta pela tabela 3.1 e que a política de controlo de acesso definida corresponde à definida por q_{ill1}. Tendo o utilizador executado a query q_1, teve acesso ao tuplo sensível (EmployeeID, LastName, FirstName) = (1, 'Davolio', 'Nancy'). Se posteriormente executou a query q_2, consegue associar o valor de Salary ao tuplo sensível previamente obtido. Segundo a abordagem baseada em grafos supramencionada, esta associação será feita sob a forma apresentada na figura 3.2.

```
1 q_ill1: select EmployeeID, LastName, FirstName from Employees where Salary >=
      2500.00;
2
3 q1: select EmployeeID, LastName, FirstName from Employees where EmployeeID = 1;
4 q2: select EmployeeID, Salary from Employees where Country = 'USA';
```

Listagem 3.4: Definição do caso de uso do Exemplo 1

Figura 3.2: Associação Simples Concetual

3.2.1 Modelos de armazenamento de informação

No âmbito do armazenamento de informação foram desenvolvidas duas soluções recorrendo a dois paradigmas. A primeira, baseada em grafos, é descrita na seção `Solução de armazenamento de informação recorrendo a grafos` do Anexo `B`. A segunda, a solução utilizada no âmbito da versão final deste trabalho, corresponde a uma evolução natural da primeira solução desenvolvida e é apresentada na subseção 3.2.1.

Solução recorrendo a Hipergrafos

A solução ao problema do armazenamento da informação resultante da execução das queries utilizando hipergrafos foi obtida como uma evolução da primeira solução utilizando grafos. A diferença entre o primeiro e o segundo paradigmas consiste no número de nós que as arestas conetam: nos grafos as arestas conetam dois nós distintos (ou *começam* e *acabam* no mesmo nó), enquanto que em hipergrafos as arestas podem conetar mais do que dois nós.

Esta abordagem surgiu no âmbito da segunda solução ao `problema do contexto` descrito na seção `Solução de armazenamento de informação recorrendo a grafos` do Anexo `B`[2], evidenciado quando uma query inquire apenas um atributo.

De fato, em [58] foi apresentado um `Modelo Universal de Estruturação dos Conhecimentos` que deve estar apto a representar quatro componentes fundamentais:

1. conjunto
2. elemento
3. relação
4. propriedade

cujo modelo gráfico de estruturação, segundo o autor, que "vem mais naturalmente ao espírito é o de Hipergrafo"[59]. Segundo este modelo cada um destes componentes são representados pelo elemento do hipergrafo tal como é apresentado na tabela 3.3

Conceito	Representação
conjunto	aresta de um hipergrafo
elemento	vértice de um hipergrafo
relação	arco juntando os vértices
propriedade	validação trazida pelo vértice

Tabela 3.3: Representação dos componentes em hipergrafo[59]

Comparativamente à solução de armazenamento de informação baseada em grafos, descrita no `Anexo B`, esta mantém a grande vantagem da mesma, que corresponde ao grau de complexidade que a operação de achar os vizinhos de um determinado nó tem. Além disto, tem a grande vantagem em relação à primeira solução ao permitir que um tuplo seja armazenado na base de dados utilizando apenas uma aresta.

Posto isto, relativamente à primeira solução, recorreu-se à mesma estratégia para armazenar os valores de atributos, quando na cláusula where da query executada existe uma condição de igualdade.

De forma a resolver o `problema do contexto`, usou-se a estratégia base da segunda abordagem proposta, que recorria à utilização de um nó de query que era posteriormente conetado aos nós dos atributos retornados pela mesma (tal como representado na figura B.8, apresentada no `Anexo B`). Perante esta solução, as restrições que indicavam o contexto em que

[2]que consiste em saber o contexto em que cada tuplo é obtido

um determinado tuplo era obtido (restrição da cláusula where e de tuplo), deixaram de ser necessárias. Posto isto, perante a execução da query q_1 a topologia do grafo criado segundo este novo paradigma será equivalente ao apresentado na figura 3.3.

```
q1: select Orders.OrderID, Orders.OrderDate, Customers.ContactName from Orders
    join Customers on Orders.CustomerID = Customer.CustomerID;
```

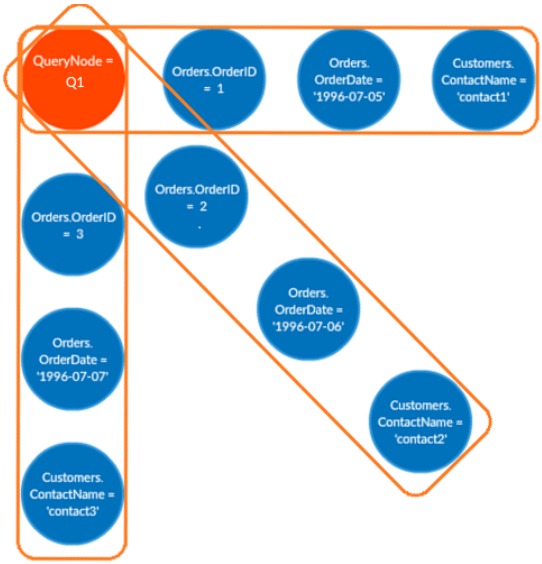

Figura 3.3: Exemplo do armazenamento de informação utilizando Hipergrafo

Adicionalmente, como forma de auxiliar o processo avaliação, que será descrito na subseção 3.3, preencheu-se a base de dados auxiliar com uma malha que contém toda a informação existente na base de dados original. A topologia desta malha corresponde à já descrita. Neste caso, o conceito de nó de query é reutilizado, sendo atribuído um a cada tabela existente na base de dados. Convém realçar que no âmbito da criação desta malha a informação obtida é proveniente das tabelas, não sendo para tal utilizadas as vistas existentes.

Tendo como exemplo o esquema da base de dados *Northwind* A.1, uma porção da malha criada é representada na figura 3.4

Posto isto, ao ser executada a query q_2, parte da topologia do grafo coincidente com a apresentada na figura 3.4, é apresentada na figura 3.5

```
q2: select OrderID, EmployeeID, CustomerID from Orders;
```

3.3 Processo de Avaliação

O processo de avaliação é repetido para cada tuplo sensível. Neste contexto podem ocorrer três resultados possíveis: o utilizador saber que se trata de um tuplo sensível protegido, tuplo sensível não protegido ou o utilizador não ter tido acesso a informação suficiente para aferir corretamente a especificação do tuplo sensível. A ocorrência do primeiro cenário simboliza o fato de um utilizador já ter efetuado queries que o colocam numa situação irregular perante as políticas de controlo de acesso definidas pelo administrador.

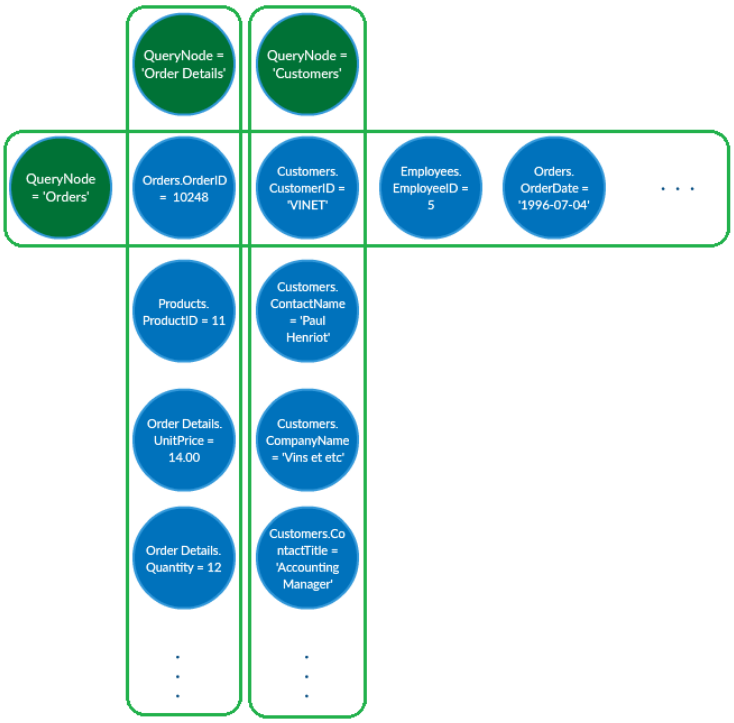

Figura 3.4: Exemplo de malha contendo toda a informação da base de dados original

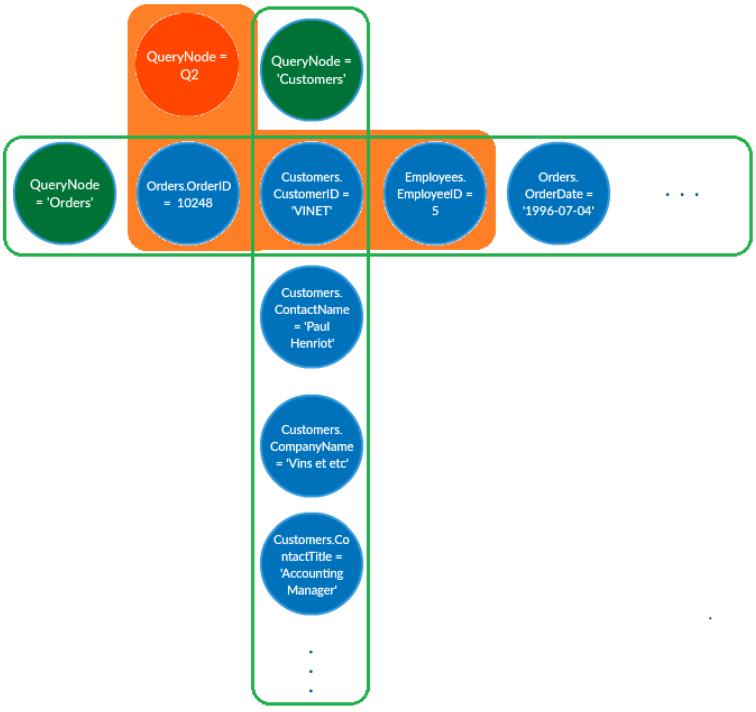

Figura 3.5: Exemplo da topologia do resultado de uma query sobreposta à malha inicial

Este processo de avaliação pode ser dividido em duas fases distintas: a primeira onde é

obtida a informação passível de ser inferida por um utilizador, e a segunda onde é avaliado se com a informação inferida, o utilizador se encontra em irregularidade perante as políticas estabelecidas.

A abordagem à primeira fase tenta ser uma aproximação ao processo de inferência a que um utilizador *malicioso* tipicamente recorre para obter informação protegida pelas políticas de acesso. Esta fase será descrita na subseção 3.3.1. A fase de avaliação da informação inferida, é descrita na subseção 3.3.2.

3.3.1 Inferência de Informação

No âmbito deste trabalho abordaram-se duas técnicas a que um utilizador *malicioso* pode recorrer para inferir informação protegida pelas políticas de controlo de acesso definidas.

A primeira corresponde à técnica de inferência baseada em *linking attack*. Ir-se-á explicar o princípio base deste tipo de ataques e, aplicando-os no âmbito desta solução, explicar como concetualmente se consegue detetá-los através da teoria de grafos.

A outra técnica de inferência baseia-se na utilização das cláusulas where das queries executadas.

Começar-se-á por explicar como se pretende resolver o problema da inferência de informação recorrendo a grafos em 3.3.1. Neste contexto, serão apresentados os conceitos e as regras básicas no processo de atravessamento do grafo em `Travessia do Grafo`. Os fundamentos da técnica de inferência baseada em cláusulas where serão descritos na subseção `Inferência de informação baseada em cláusulas where`. Por fim, será apresentada a abordagem ao processamento entre as técnicas de inferência de informação na subseção `Agregação dos resultados obtidos`.

Inferência de informação baseada em grafos

No `Exemplo 1` apresentado no âmbito da listagem 3.4, pôde-se perceber o conceito base desta técnica de inferência aplicado à solução desenvolvida no âmbito desta dissertação. Este primeiro exemplo foi extremamente simples, tendo conseguido obter o valor do atributo sensível de predicado através de uma única associação. No âmbito deste modelo, esta associação é simbolizada na ação de atravessar a aresta com início no nó correspondente ao atributo EmployeeID e com fim no nó correspondente ao atributo Salary. No entanto, é expetável que a inferência de valores de atributos sensíveis de predicado possa ser realizada através de associações sucessivas. Este cenário é então simbolizado pelo atravessamento de várias arestas no grafo.

Serão agora apresentadas duas possíveis estratégias que permitem atravessar múltiplas arestas de forma a conseguir chegar aos valores de atributos sensíveis de predicado a partir dos valores que compõem o tuplo sensível.

Busca exaustiva em profundidade

A primeira abordagem consiste na utilização da estratégia busca exaustiva em profundidade, descrita na subseção 2.2.1, onde se parte de cada nó pertencente ao tuplo sensível e se efetua uma busca em profundidade sobre todos os nós válidos no contexto do mesmo.

Tendo em conta o esquema da base de dados *Northwind*, apresentado na figura A.1, suponha-se que é definida a política de controlo de acesso representada por q_{ill} e que, adicio-

nalmente, são executadas as queries apresentadas na listagem 3.5.

```
1 q_ill: select CustomerID, CompanyName, ContactName from Customers join Orders on
      Customers.CustomerID = Orders.CustomerID join OrderDetails on Orders.OrderID
      = OrderDetails.OrderID and UnitPrice > 50.00;
2
3 q1: select OrderID, EmployeeID, CustomerID from Orders;
4 q2: select EmployeeID, FirstName, LastName from Employees;
5 q3: select CustomerID, CompanyName, ContactName from Customers;
6 q4: select Salary, Country from Employees;
7 q5: select LastName, FirstName, Country from Employees;
8 q6: select OrderID, UnitPrice from OrderDetails;
```

Listagem 3.5: Definição do caso de uso estudado no âmbito dos algoritmos de busca

Neste caso, a topologia do grafo criado pelas queries executadas corresponde à apresentada na figura 3.6. De notar que na realidade cada *Customer* está associado a múltiplas *Orders*, cada *Order* está associada a múltiplos *UnitPrice*, bem como cada *FirstName* e *LastName* podem estar associados a múltiplos *Country*. Apesar desta representação não estar totalmente correta, foi a usada por motivos de simplificação.

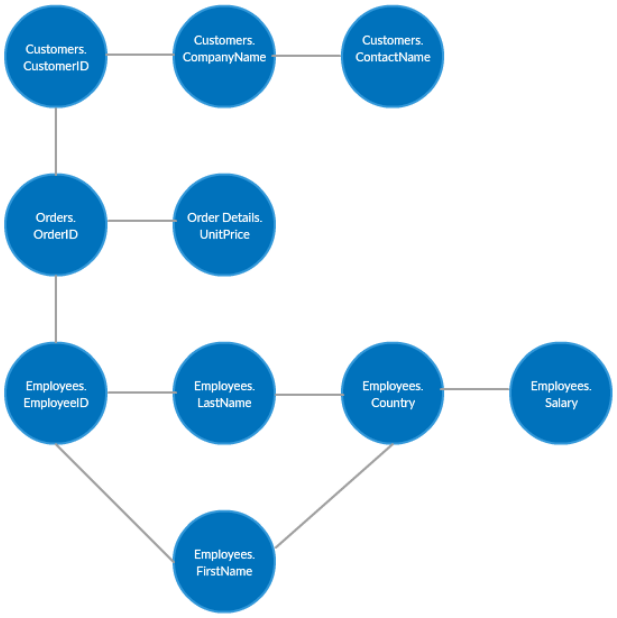

Figura 3.6: Exemplo da topologia do grafo criado

Neste caso a abordagem baseada na pesquisa em profundidade começa por, numa primeira iteração a partir de cada nó que constitui o tuplo sensível, tentar obter uma aresta para outro nó válido. Nesta iteração, os nós dos quais se devem partir aparecem representados pelos quadrados vermelhos na figura 3.7. No âmbito desta abordagem a pesquisa é efetuada recursivamente até não encontrar mais nós válidos no âmbito dos valores já obtidos ou até obter nós de UnitPrice.

Perante este exemplo, pode-se facilmente perceber que os nós correspondentes aos atributos ContactName e CompanyName, não possuem nenhuma conexão. Posto isto, a partir do

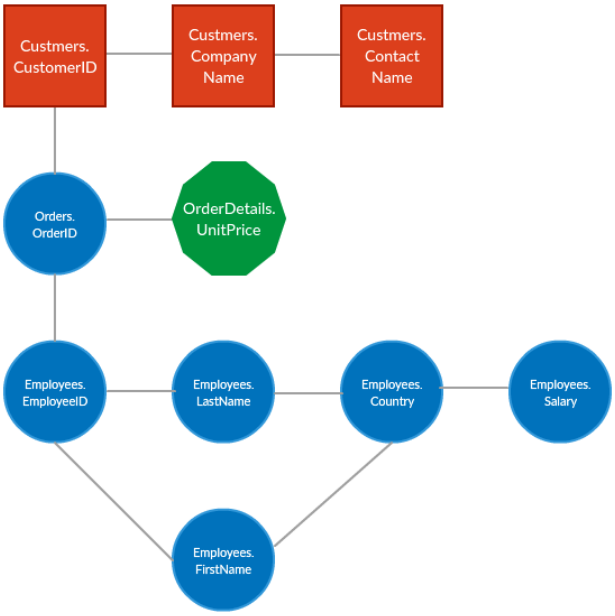

Figura 3.7: Exemplo da topologia - primeira iteração da pesquisa em profundidade

nó CustomerID consegue-se avançar para o conjunto de nós {OrderID, EmployeeID}, visto que são retornados juntamente com CustomerID através de q_1. A respetiva representação é apresentada na figura 3.8. A partir destes nós conseguem-se obter os valores de UnitPrice associados ao nó OrderID. Adicionalmente, a partir do nó EmployeeID esta abordagem continua a busca em profundidade, apesar de neste exemplo, posteriormente, não serem obtidos mais valores do atributo sensível de predicado UnitPrice associados ao tuplo sensível.

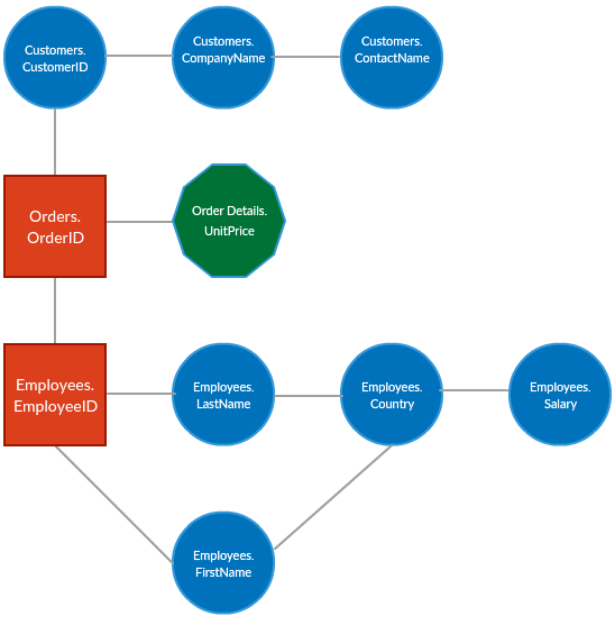

Figura 3.8: Exemplo da topologia - segunda iteração da pesquisa em profundidade

Pesquisa Orientada

Outra forma de obter inferir informação através do grafo é através de uma pesquisa orientada. Como se pôde constatar na subseção anterior, a abordagem descrita era ineficiente, relativamente aos nós visitados ao longo do processo de varrimento do grafo. Esta abordagem, corresponde a uma versão otimizada da primeira e à solução adotada no âmbito desta dissertação.

Ao conhecer a topologia do grafo é possível tirar partido desse conhecimento para tornar a pesquisa de informação mais eficiente. A utilização deste conhecimento permite que quando se está a percorrer o grafo não se efetuem uma pesquisa em profundidade, que no limite pode levar a que seja necessário visitar todos os nós do grafo.

Surge então a noção de `Caminho`: corresponde à projeção de atributos inquiridos entre múltiplas queries, conetando os atributos que constituem o tuplo sensível aos atributos sensíveis de predicado. Convém realçar que este processamento, interpreta apenas o esquema dos atributos inquiridos em cada query, ignorando para tal o valor dos atributos efetivamente inquiridos.

Posto isto, no âmbito do exemplo apresentado na listagem 3.5, através da análise das queries, pode-se perceber que apenas se precisa de percorrer as arestas resultantes da execução das queries q_1 e q_6 para obter os possíveis valores de UnitPrice associados a cada Customer. A representação desta topologia é apresentada na figura 3.9.

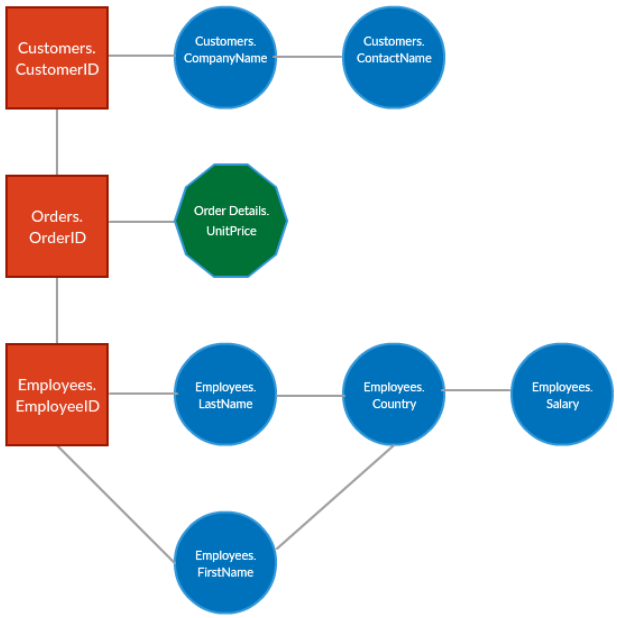

Figura 3.9: Exemplo da topologia - pesquisa orientada

Travessia do Grafo

Tal como já foi mencionado, o fato de esta solução ter conseguido atravessar o grafo desde um dos nós que compõem o atributo sensível até aos nós que representam os atributos sensíveis de predicado simboliza o fato de ser possível, com as queries já executadas, a um utilizador inferir informação crítica.

Até este momento, nos exemplos apresentados, as projeções em atributos comuns às queries através das quais é possível percorrer o grafo desde o nó inicial até ao final incidiram sobre as chaves primárias das tabelas *atravessadas*. Adicionalmente, ainda não foi apresentado nenhum exemplo onde a cláusula where de uma query atravessada não fosse nula.

Posto isto, é necessário ter alguns cuidados no processo de atravessamento do grafo. A cada aresta percorrida é necessário avaliar se a restrição da cláusula where da query associada à respetiva aresta é válida no contexto do tuplo sensível e dos valores dos atributos obtidos até então. Neste caso, pode-se então atravessar a aresta se a restrição da cláusula where for válida no respetivo contexto.

Além da avaliação da cláusula where, também é necessário verificar se se pode descartar algum tuplo que seja inválido no contexto atual e que, através das queries já executadas, um utilizador consiga aferir a sua invalidade. Atendendo a um exemplo em que o esquema da base de dados corresponde ao apresentado na figura A.1, que foi definida uma política de controlo de acesso através da query q_{ill} e que um utilizador executou as queries apresentadas na listagem 3.6.

```
1 q_ill: select CustomerID, CompanyName, ContactName from Customers join Orders on
    Customers.CustomerID = Orders.CustomerID join OrderDetails on Orders.OrderID
    = OrderDetails.OrderID and UnitPrice > 50.00;
2
3 q1: select CustomerID, CompanyName, ContactName from Customers;
4 q2: select OrderID, EmployeeID, CustomerID from Orders;
5 q3: select CustomerID from Orders where OrderDate > '1997-01-01';
6 q4: select EmployeeID, OrderDate, UnitPrice from Orders join OrderDetails on
    Orders.OrderID = OrderDetails.OrderID;
```

Listagem 3.6: Definição do caso de uso estudado no âmbito da travessia do grafo

Ao partir do tuplo sensível, obtém-se as *Orders* associadas ao mesmo através de q_2. Posteriormente, suponha-se que o CustomerID do tuplo sensível que se está a avaliar não é retornado por q_3, o que significa que a respetiva compra foi efetuada antes de 1997-01-01. Suponha-se também que o mesmo EmployeeID está associado a outra compra (OrderID) que foi efetuada depois da data supramencionada. Ao atravessar o grafo nas arestas que conetam EmployeeID a UnitPrice (e também OrderDate) deve-se descartar os tuplos que obedecem à condição C_1 = (OrderDate > '1997-01-01'), visto que o utilizador sabe que respetivo Customer não está associado a C_1.

Até agora, para atravessar o grafo apenas foi considerado o caso de existir um *caminho* entre os nós do tuplo sensível e os nós dos atributos sensíveis de predicado. No entanto também é possível inferir informação quando não existe um caminho direto entre os nós supramencionados. De fato, tal como é apresentado no exemplo baseado na listagem 3.3 da subseção 3.1.1, verifica-se que é possível obter um conjunto de possíveis valores de Salary para os empregados com (EmployeeID > 4). Estes casos podem ser percebidos como caminhos de passos únicos, na medida em que são compostos apenas pela query na qual são inquiridos os valores dos atributos sensíveis de predicado.

Inferência de informação baseada em cláusulas where

Na subseção anterior foi apresentada uma estratégia de inferência de informação recorrendo a projeções entre atributos comuns de múltiplas queries, que no âmbito da desta solução

são simbolizados por um atravessamento do grafo. A estratégia apresentada consistia em recorrer apenas aos atributos presentes na cláusula select de cada query. Além da informação retornada por uma query, muitas vezes a cláusula where da mesma pode ser também usada para inferir informação.

Através do exemplo apresentado na listagem 3.6 percebeu-se que através da cláusula where de q_3 seria possível a um utilizador inferir informação que o ajudasse a descartar alguns tuplos retornados por q_4 e, consequentemente, possíveis valores de UnitPrice associados a um dado tuplo sensível.

Como será fácil de perceber, além de poder ser utilizada para a filtragem de valores de atributos obtidos através da técnica de inferência anterior, podem-se também obter as restrições através desta técnica de inferência e avaliá-las diretamente no âmbito do processo de avaliação. Considere-se o estado da base de dados apresentado na tabela 3.4 e que um utilizador tem permissão para executar as queries apresentadas na listagem 3.7.

EmployeeID	LastName	FirstName	Country	Salary
1	Davolio	Nancy	USA	2954.55
2	Fuller	Andrew	USA	2254.49
3	Peacock	Margaret	USA	1861.08
4	Dodsworth	Nancy	USA	3119.15
5	Buchanan	Steven	UK	1744.21
6	Suyama	Michael	UK	2004.07
7	King	Robert	UK	1991.55
8	Callahan	Laura	USA	2100.5
9	Dodsworth	Nancy	UK	1333.33

Tabela 3.4: Exemplo da tabela Employees (3)

```
1 q_ill: select EmployeeID, FirstName, LastName from Employees where Salary > 2000;
2
3 q_1: select EmployeeID, FirstName, LastName from Employees;
4 q_2: select EmployeeID from Employees where Salary >= 2500;
```

Listagem 3.7: Definição de caso de uso recorrendo a cláusulas where (1)

Ao conseguir associar os tuplos sensíveis identificados por { EmployeeID = 1; EmployeeID = 4} à restrição apresentada na cláusula where de q_2, mesmo sem obter valores concretos de Salary aos quais estão associados, consegue-se inferir que estes tuplos correspondem a tuplos sensíveis protegidos, violando assim a política de controlo de acesso estabelecida.

Agregação dos resultados obtidos

Quando se obtém informação sobre um atributo sensível de predicado através de mais de um caminho (apresentados na subseção `Pesquisa Orientada`) e/ou uma cláusula where (apresentado na subseção `Inferência de informação baseada em cláusulas where`) torna-se necessário efetuar um processamento com o intuito de reduzir os possíveis valores de atributos sensíveis de predicado associados ao tuplo sensível que se está a avaliar.

De fato, se analisar o exemplo apresentado na listagem 3.6, percebe-se que é possível limitar o conjunto de valores válidos de OrderDate obtidos em q_4 através da informação da

cláusula where de q_3. Este processamento será descrito em detalhe na subseção `Filtragem de valores obtidos através de um caminho` 4.7.4.

Além do exemplo supramencionado, pode-se também efetuar este processamento entre múltiplos Caminhos e entre informação inferida entre cláusulas where de múltiplas queries.

Tendo em conta o estado da base de dados apresentado na tabela 3.4, e o caso de uso apresentado na listagem 3.8, para o empregado identificado por (EmployeeID = 9) pode-se obter o valor de Salary associado ao mesmo através da operação de interseção entre os resultados obtidos através dos caminhos formados pelas queries $C_1 = \{q_1, q_4\}$ e $C_2 = \{q_1, q_2, q_3\}$ apresentados nas tabelas 3.5 da esquerda e direita, respetivamente.

```
q1: select EmployeeID, FirstName, LastName from Employees;
q2: select LastName, FirstName, Country from Employees;
q3: select Salary, Country from Employees;
q4: select FirstName, Salary from Employees;
```

Listagem 3.8: Definição de caso de uso com vista a agregação de múltiplos conjuntos de resultados obtidos (1)

Salary
2954.55
3119.15
1333.33

Salary
1744.21
2004.07
1991.55
1333.33

Tabela 3.5: Salários inferidos através dos Caminhos C_1 e C_2

No âmbito dos exemplos apresentados até este momento, as operações de processamento entre as múltiplas ocorrências das técnicas de inferência de informação consistiram na interseção da informação obtida.

No entanto nem sempre é possível realizar esta operação; isto pode dever-se ao fato de os valores dos atributos sensíveis de predicado serem obtidos associados a contextos distintos (cuja interseção é nula), como é exemplificado no exemplo definido pelas queries apresentadas na listagem 3.9. Considerem-se os possíveis caminhos formados pelas queries $C_{1.1} = \{q_1, q_2, q_3\}$ e $C_{2.1} = \{q_1, q_2, q_4\}$. Através da análise ao contexto em que as queries q_3 e q_4 obtêm os resultados, pode-se perceber que não se intersetam.

```
qill: select CustomerID, CompanyName, ContactName from Customers join Orders on
    Customers.CustomerID = Orders.CustomerID join OrderDetails on Orders.OrderID
    = OrderDetails.OrderID and UnitPrice > 50.00;

q1: select CustomerID, CompanyName, ContactName from Customers;
q2: select OrderID, EmployeeID, CustomerID from Orders;
q3: select EmployeeID, UnitPrice from Orders join OrderDetails on Orders.OrderID
    = OrderDetails where ProductID <= 5;
q4: select EmployeeID, UnitPrice from Orders join OrderDetails on Orders.OrderID
    = OrderDetails where ProductID > 10;
```

Listagem 3.9: Definição de caso de uso com vista a agregação de múltiplos conjuntos de resultados obtidos (2)

Além de os valores dos atributos sensíveis de predicado poderem ser obtidos em contextos distintos, pode também ocorrer o caso de estarem associados ao mesmo contexto (ou parte dele) sem que, no entanto, o utilizador tenha informação suficiente para efetuar essa associação. Este caso é apresentado na listagem 3.10. Considerem-se os possíveis caminhos formados pelas queries $C_{1.2} = \{q_1, q_2, q_3\}$ e $C_{2.2} = \{q_1, q_2, q_4\}$. Através da análise ao contexto em que as queries q_3 e q_4 obtêm os resultados, e da informação à qual se tem acesso, pode-se perceber que a mesma não permite saber quais os contextos que se intersetam, exceto para as situações identificadas na subseção 3.3.1: valores de EmployeeID que estejam associados a uma única compra ou que todas as compras a que um dado EmployeeID está associado estejam associadas às condições (ProductID <= 5) e (Quantity > 4).

```
1 q_ill5: select CustomerID, CompanyName, ContactName from Customers join Orders on
      Customers.CustomerID = Orders.CustomerID join OrderDetails on Orders.OrderID
      = OrderDetails.OrderID and UnitPrice > 50.00;
2
3 q1: select CustomerID, CompanyName, ContactName from Customers;
4 q2: select OrderID, EmployeeID, CustomerID from Orders;
5 q3: select EmployeeID, UnitPrice from Orders join OrderDetails on Orders.OrderID
      = OrderDetails where ProductID <= 5;
6 q4: select EmployeeID, UnitPrice from Orders join OrderDetails on Orders.OrderID
      = OrderDetails where Quantity > 4;
```

Listagem 3.10: Definição de caso de uso com vista a agregação de múltiplos conjuntos de resultados obtidos (3)

Perante esta incapacidade de obter a interseção dos resultados obtidos, de forma a não os *perder* deve-se agrupá-los utilizando uma operação de união.

Por fim, em alguns casos é possível utilizar a operação de diferença entre os resultados obtidos. Este caso é exemplificado no âmbito da listagem 3.11. Para o contexto da base de dados apresentado na tabela 3.4, caso o tuplo sensível que se esteja a avaliar seja identificado por (EmployeeID = 1), pode-se restringir o conjunto de possíveis valores de Salary associados ao mesmo através da operação de diferença obtida entre os resultados obtidos entre $C_{1.3} = \{q_1, q_3\}$ e $C_{2.3} = \{q_1, q_2\}$.

```
1 q_ill1: select EmployeeID, LastName, FirstName from Employees where Salary >=
      2500.00;
2
3 q1: select EmployeeID, FirstName, LastName from Employees;
4 q2: select Salary from Employees where EmployeeID > 4;
5 q3: select FirstName, Salary from Employees;
```

Listagem 3.11: Definição de caso de uso com vista a agregação de múltiplos conjuntos de resultados obtidos (4)

Na subseção 4.7.5 será abordada a sua implementação, onde serão aprofundados alguns detalhes da mesma.

3.3.2 Avaliação dos resultados obtidos

Na subseção 3.3.1, foi apresentado o modelo para obter a informação passível de ser inferida através das queries já executadas por um utilizador. Nesta subseção será descrito o modelo de

avaliação, recorrendo à informação supramencionada. Esta solução avalia os valores obtidos dos atributos sensíveis de predicado no contexto das cláusulas where das respetivas q_{ill}.

Posto isto, convém definir os tipos de resultados que podem surgir da subseção 3.3.1. Os resultados provenientes dos caminhos obtidos através da projeção do esquema de atributos inquiridos em múltiplas queries, ou seja, quando se obtêm valores de atributos efetivos, podem ser divididos em duas categorias:

- no conjunto de valores obtidos, todos estão associados ao tuplo sensível que se está a avaliar, tal como é apresentado no exemplo definido na listagem 3.5;
- no conjunto de valores obtidos, nem todos estão associados ao tuplo sensível que se está a avaliar, tal como é apresentado no exemplo definido na listagem 3.6.

Tipicamente, o primeiro caso ocorre quando os atributos projetados entre as queries que formam o caminho, correspondem às chaves das tabelas atravessadas. Em contrapartida, quando as projeções dos atributos entre as queries que formam o caminho correspondem a atributos comuns, os resultados obtidos podem não estar todos associados ao tuplo sensível.

Tal como na `lógica booleana` [60], pode-se considerar que a expressão $A \wedge B$ é verdadeira quando ambos os valores de A e B são verdadeiros, enquanto que na expressão $C \vee D$, basta que apenas um dos acontecimentos C ou D seja verdadeiro para que a expressão também o seja.

Analogamente, quando se avalia um conjunto de valores que são todos associados ao tuplo sensível, basta que apenas um seja válido no contexto da cláusula where para que se possa concluir que o mesmo é válido.

Por outro lado, se nem todos os valores obtidos forem associados ao tuplo sensível, para considerar que o tuplo sensível é válido no contexto de q_{ill}, todas as combinações de valores têm que o ser. No exemplo definido na listagem 3.11, consegue-se inferir que o empregado idenficado por (EmployeeID = 1), corresponde a um tuplo sensível protegido pois o Resultado (RS) de $\{C_{1.3}$ - $C_{2.3}\}$ = {3119.15, 2954.55}. Os possíveis valores de Salary associados ao respetivo tuplo são ambos válidos no âmbito da cláusula where de q_{ill1} (Salary >= 2500).

Adicionalmente, tal como apresentado na seção 3.3.1, é possível inferir informação através da cláusula where das queries executadas. Seja R_{ill} a restrição da cláusula where de q_{ill} e R_i a restrição da cláusula where de q_i.

Perante este cenário, é possível inferir que um tuplo sensível é protegido quando se associá-lo ao tuplo retornado por uma query q_i, cuja $R_i \subset R_{ill}$. Considere-se que R_{ill} = {Salary > 2000} e R_i = {Salary > 2500}. Se se conseguir associar o tuplo sensível a um tuplo retornado por q_i, pode-se concluir que o tuplo está associado a (Salary > 2500 > 2000).

Alternativamente, considera-se que um tuplo sensível é não protegido quando não está associado a nenhum tuplo retornado por uma query q_i, cuja $R_{ill} \subset R_i$. Considere-se que R_{ill} = {Salary > 2500} e R_i = {Salary > 2000}. Se o tuplo sensível não estiver associado a nenhum tuplo retornado por q_i, pode-se concluir que o tuplo está associado a (Salary <= 2000 < 2500).

Capítulo 4

Prova de Conceito

Neste capítulo será descrita a implementação da solução apresentada no capítulo 3. De fato, tal como se pode adivinhar através da leitura do respetivo capitulo, foi implementada parcialmente uma primeira solução, recorrendo ao paradigma de grafos e, na segunda, recorrendo ao paradigma de hipergrafos, que corresponde a uma evolução natural da solução ao problema proposto.

Começar-se-á por apresentar a arquitetura geral da solução desenvolvida na seção 4.1. De seguida descrever-se-á a implementação dos componentes principais desta solução.

- A seção 4.2 apresenta o `Desenvolvimento do Orchestrator`;
- A seção 4.3 apresenta o `Desenvolvimento do Schema Interpreter`;
- A seção 4.4 apresenta o `Desenvolvimento do Query Interpreter`;
- A seção 4.5 apresenta o `Desenvolvimento do Query Executor`;
- A seção 4.6 apresenta o `Gestor de Políticas`;
- A seção 4.7 apresenta o `Desenvolvimento do Evaluator`.

Por fim será explicada a forma como é possível a um administrador de sistemas utilizar a plataforma desenvolvida no âmbito desta dissertação, na seção 4.8 e em 4.9 será definido o caso de uso com o qual se pretende avaliar o trabalho desenvolvido e discutidos alguns dos resultados obtidos nesse contexto.

No âmbito do desenvolvimento desta solução foi necessária a criação de estruturas auxiliares. No Anexo `C` são apresentadas as unidades básicas de armazenamento de informação. Nele é explicada a importância das mesmas e as operações que se podem realizar ao nível de cada uma.

4.1 Arquitetura Geral

A figura 4.1 apresenta a arquitetura da solução desenvolvida no âmbito desta dissertação.

O `Orchestrator` corresponde ao componente desta solução que serve de interface para aplicações terceiras que a pretendam utilizar. No âmbito da execução de uma query, este

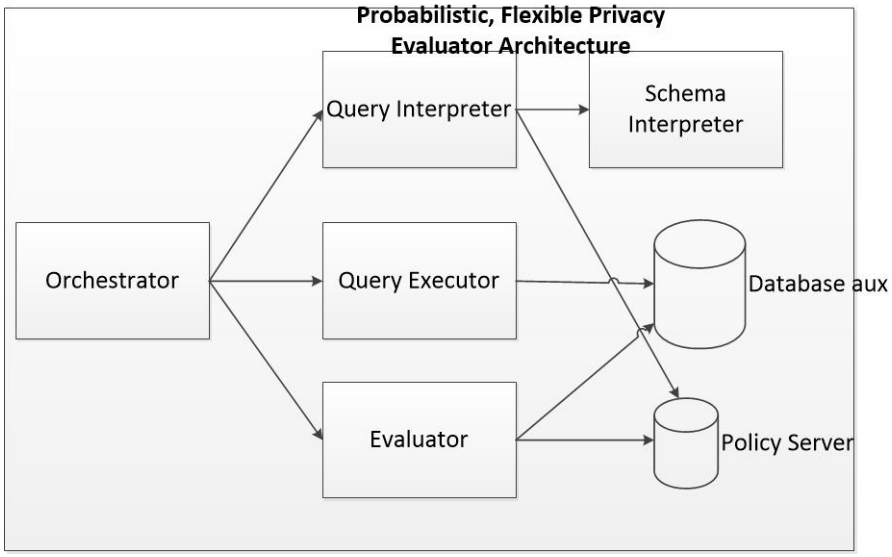

Figura 4.1: Arquitetura PFPE

componente é responsável por (1) processá-la recorrendo ao `Query Interpreter`, (2) executá-la recorrendo ao `Query Executor` e, por fim, (3) avaliar se a informação passível de ser inferida coloca um utilizador *malicioso* em posição quebrar as políticas de controlo de acesso probabilísticas definidas recorrendo ao `Evaluator`.

No âmbito do processo de interpretação de queries, realizado em `Query Interpreter`, é obtida uma instância da estrutura do tipo de dados `Query` (desenvolvida nesta dissertação) que armazena os atributos e tabelas inquiridos bem como o contexto em que os respetivos valores serão obtidos (tipicamente obtido através da cláusula where da mesma). Este componente recorre ao `Schema Interpreter` de forma a conseguir perceber quais os atributos e as tabelas presentes nas declarações SQL submetidas pelo programador da aplicação. Neste contexto, pode ocorrer o caso de ser necessário uma transformação adicional dependendo dos atributos inquiridos pela query e dos atributos que não devem ser utilizador no âmbito do processo de avaliação, apresentados na subseção 3.1.2, e que se encontram armazenados em `Policy Server`. Este componente além de armazenar este conjunto de atributos, armazena também as políticas de controlo de acesso probabilísticas.

Tal como já referido, o `Query Executor` é responsável pela execução das queries submetidas e o respetivo armazenamento em `Database aux`, sob a forma apresentada na seção 3.2.

Relativamente ao processo de avaliação, realizado pelo componente `Evaluator`, é verificado para cada tuplo sensível de cada política de controlo de acesso probabilística estabelecida se um utilizador malicioso, através da informação já obtida (armazenada em `Database aux`) consegue violar as políticas supramencionadas. No âmbito deste processo é produzido um relatório que especifica através de quais queries e técnicas de inferência um utilizador consegue inferir informação dos atributos sensíveis de predicado associados a cada tuplo sensível.

4.2 Desenvolvimento do `Orchestrator`

No âmbito desta solução, o componente `Orchestrator` é apenas responsável por delegar as tarefas aos restantes componentes de alto nível, tal como apresentado na figura 4.1. A sequência de tarefas executadas por cada query submetida à aplicação é apresentada no algoritmo 3. De fato, ao nível da execução da query é verificado se já não foi executada uma query igual (que inquire os mesmos atributos no mesmo contexto). Adicionalmente, tal como referido no capítulo 1, no âmbito da execução de cada query, o processo de avaliação é efetuado de acordo com o que o administrador tenha definido.

```
1 foreach submitted query statement do
2 |   parse query statement through QueryInterpreter;
3 |   execute query and store its results through QueryExecutor;
4 |   foreach access policy do  evaluate information acquired through Evaluator ;
5 end
```

Algoritmo 3: Trabalho realizado pelo `Orchestrator`

4.3 Desenvolvimento do `Schema Interpreter`

Para avaliar se as políticas de controlo de acesso são cumpridas pelos múltiplos acessos à base de dados que se pretende proteger é necessário, antes de mais, conhecer o esquema da mesma. O acesso a ele é importante na medida em que é necessário saber qual a informação inquirida/retornada, aquando da execução de uma query.

Neste contexto, é importante então ter forma de obter as tabelas existentes, os seus atributos e as restrições de chaves (candidatas e estrangeiras) de cada uma. Por exemplo, tendo em conta o esquema da base de dados A.1 apresentada no Anexo `A`, ao executar as queries q_1 e q_2, definidas na listagem 4.1

```
1 Q1: select OrderID, EmployeeID, CustomerID from Orders;
2 Q2: select EmployeeID, FirstName, LastName from Employees;
```

Listagem 4.1: Definição de queries a serem interpretadas

é importante ter a capacidade de perceber que o atributo EmployeeID inquirido na query q_1 corresponde ao mesmo inquirido em q_2, apesar de serem inquiridas tabelas distintas.

Neste contexto, é importante então ter forma de obter as tabelas existentes: os seus atributos e as restrições de chaves (primárias e estrangeiras) de cada uma. Recorrendo a uma implementação da interface DatabaseMetaData do JDBC é possível obter os nomes de todas as tabelas através do método `getTables`. Posteriormente, para cada tabela é possível obter os respetivos atributos e tipos de dados através do método `getColumns`. Para a obtenção as restrições de chaves únicas é utilizado o método `getPrimaryKeys` e as chaves estrangeiras o método `getImportedKeys`.

Além das tabelas, podem também ser definidas vistas no âmbito do esquema de uma base de dados. A elas estão normalmente associadas limitações adicionais que devem ser tidas em conta: as cláusulas where utilizadas na sua criação, para limitar a informação associada às mesmas. Por exemplo, a vista definida por

```
1 V1: create view Top_Employees as select * from Employees where Salary >
    2500.00;
```

Listagem 4.2: Exemplo Criação de Vista Top Employees

seleciona apenas os empregados que possuam um salário superior a 2500. No entanto, através da utilização da implementação da interface DatabaseMetaData não é possível ter acesso a estas restrições. Apesar disto, pode-se também perceber que a sintaxe das declarações `create view` e `select` são bastante similares. Posto isto, com o devido processamento da declaração `create view` é possível obter o esquema das vistas, recorrendo à solução descrita na seção `Interpretação de Queries` 4.4. Este processamento consiste em descartar o início da declaração de forma a ficar apenas com uma declaração do tipo `select`; ou seja, no caso da vista definida na listagem 4.2, deve-se apenas obter a declaração `select * from Employees where Salary > 2500.00`.

4.3.1 Estrutura DatabaseSchema

A informação sobre o esquema da base de dados é armazenada na estrutura `Database-Schema`.

Os dados efetivamente armazenados correspondem apenas ao conjunto das tabelas, onde é armazenado o nome e o tipo de dados de cada atributo pertencente a cada uma das tabelas e as restrições relativas às mesmas:

- restrições de chaves únicas;
- restrições de chaves estrangeiras;
- ao nível das vistas, caso se aplique, a restrição ao nível do contexto em que a informação das mesmas é obtida.

A verdadeira relevância desta estrutura no âmbito do trabalho desenvolvido prende-se com as operações realizadas ao nível da mesma. A título de exemplo, (1) a relação entre as tabelas, (2) a obtenção de um caminho entre duas tabelas, (3) a normalização dos nomes dos atributos, entre outras, são operações realizadas no contexto desta estrutura.

4.4 Desenvolvimento do `Query Interpreter`

O processamento de queries SQL corresponde a um requisito bastante importante no âmbito da solução desenvolvida nesta dissertação. No âmbito da interpretação das mesmas, recorreu-se à ferramenta `FoundationDB`[6]. O Anexo `D` apresenta os princípios base do processamento de uma query recorrendo à ferramenta `FoundationDB`.

Após obter a informação da query é necessário efetuar uma normalização de cada um dos seus componentes. Esta padronização é efetuada recorrendo à estrutura `Database- Schema`, apresentada em 4.3.1. No âmbito desta normalização é necessário obter os nomes dos atributos sob a forma <table_name>.<attribute_name>, apresentada na seção 3.2 e associar os respetivos tipos de dados a cada atributo utilizado. A seção `Algoritmo de normalização de nomes de atributos` do Anexo `E` apresenta a forma como é efetuada esta padronização.

Tendo em conta que o objetivo deste trabalho não era desenvolver um *parser* para todas as possíveis queries SQL, é suportado um subconjunto limitado de queries: (1) as queries mais comuns (compostas apenas pelas cláusulas *select*, *from* e *where*), (2) com cláusulas *inner join* (onde apenas são retornados tuplos quando existe informação comum nas duas tabelas), (3) subqueries presentes nas cláusulas *from* e (4) condição *in*.

4.4.1 Tratamento de cláusulas where

Como se pode facilmente perceber, um aspeto bastante importante no âmbito desta solução corresponde à interpretação da cláusula where de uma query: fornece o contexto em que a informação da respetiva query é obtida. Considere-se a query apresentada na listagem C.1.

Pode-se verificar que apenas são retornados empregados cujo país de origem corresponda a `USA` e tenham nascido depois de `01-01-1985`. Esta informação é armazenada recorrendo à estrutura `Restriction`, apresentada na seção `Agregação de Expressões Simples` do Anexo `C`. A também já apresentada figura C.1 corresponde à sua representação.

4.4.2 Interpretação de Queries sobre Vistas

Quando um utilizador inquire uma vista, torna-se necessário adicionar à (eventual) restrição da query efetuada a (eventual) restrição utilizada no momento da definição dessa mesma vista. Neste caso se um utilizador inquirir a vista `Top_Employees` através da query q_1

```
1 Q1: select * from Top_Employees where Country = 'USA' or Country = 'UK';
```

Listagem 4.3: Definição de query sobre vista Top_Employees

a restriçao final da query q_1 é dada pela restrição apresentada na figura 4.2.

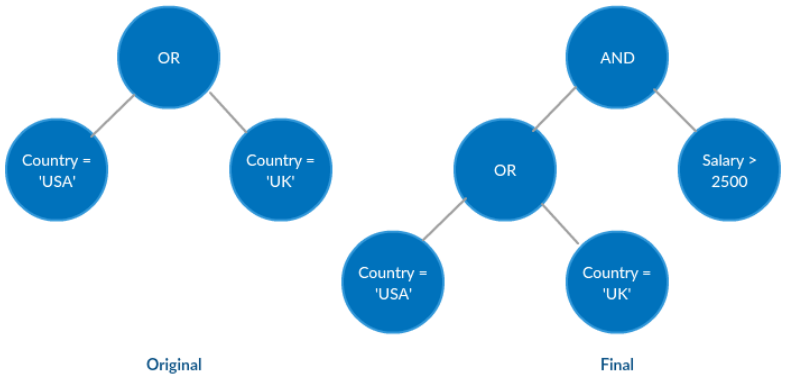

Figura 4.2: Restrição original e final de q_1

Comparando ambas as árvores, pode-se perceber que é possível adicionar esta restrição, simplesmente adicionando um nó com um operador lógico `and` e atribuir a um ramo a restrição original da query e ao outro a restrição associada á vista inquirida.

4.4.3 Transformação da cláusula select de Queries

No âmbito da interpretação de queries submetidas pelo utilizador é necessário aplicar um último nível de processamento.

Tal como mencionado na subseção 3.1.2, perante alguns cenários é necessário adicionar alguns atributos ao conjunto de atributos a serem inquiridos por uma dada query. A sua implementação, recorrendo à estrutura `Query`, torna-se bastante simples. No âmbito deste processamento, após identificar quais os atributos a adicionar, é necessário apenas acrescentá-los à estrutura `attributesQueried`, que contém os atributos inquiridos na cláusula *select* da respetiva query.

4.5 Desenvolvimento do `Query Executor`

Nesta seção será descrita a forma como é executada cada query submetida por um utilizador. A solução recorrendo ao paradigma de base de dados de grafos, que corresponde ao paradigma utilizado na primeira versão do trabalho efetuado no âmbito desta dissertação, é apresentada na seção `Execução de queries - solução baseada em grafos` do Anexo `B`. Relativamente à solução recorrendo a Hipergrafos, correspondente à abordagem utilizada na versão final do trabalho desenvolvido, a mesma é apresentada na subseção 4.5.1.

Tal como foi referido na seção 3.2, a solução baseada em hipergrafos correspondeu a uma evolução relativamente à primeira, baseada em grafos, onde foram resolvidos os problemas levantados pelo fato de as arestas em grafos apenas poderem conetar dois nós. Posto isto, deve-se ter em consideração que algumas estratégias utilizadas presentes na solução final, não foram desenvolvidas no âmbito da solução baseada em grafos.

No âmbito da execução das queries, a inicialização da base de dados auxiliar com uma malha contendo a informação total armazenada pela base de dados original, não foi implementada aquando do desenvolvimento da solução baseada em grafos. Este aspeto tem especial importância na diferença de como as queries submetidas pelo utilizador são executadas no âmbito das duas soluções.

Adicionalmente, as possíveis soluções propostas ao problema do contexto apresentadas no âmbito da seção `Solução de armazenamento de informação recorrendo a grafos` do Anexo `B`, nas quais se podia passar a restrição da cláusula where em que os nós eram obtidos para os próprios nós, tal como foi exposto na figura B.7 ou através da utilização de um nó de query, demonstrado na figura B.8, não chegaram a ser efetivamente implementadas.

4.5.1 Execução de queries - solução baseada em hipergrafos

No âmbito da abordagem baseada em hipergrafos, utilizou-se a base de dados `HyperGraphDB` para armazenar a informação inquirida por cada query.

Ao contrário da ferramenta `Neo4J`, esta possibilita o armazenamento de objetos complexos. Visto que a estrutura `SingleExpression` é utilizada para armazenar os valores dos atributos e durante o processo de avaliação, tal como será apresentado na seção 4.7, pode-se armazenar diretamente cada instância desta estrutura diretamente no grafo. Ao não precisar de decompor a estrutura para a armazenar no grafo e posteriormente reconstrui-la à medida que o mesmo é atravessado, torna esta solução mais eficiente não apenas em termos de código necessário como também de desempenho.

Tal como mencionado nesta seção, a base de dados auxiliar é inicializada com a informação total armazenada pela base de dados que se pretende proteger.

Definição dos Nós de Query

Cada query executada está associada a um nó de query único. Tal como mencionado na subseção 3.2.1, cada tuplo retornado pela respetiva query é também conetado a este nó de query.

À semelhança da forma em que os valores dos atributos são armazenados, estes nós são também armazenados numa `SingleExpression`. Neste caso, o <attribute_name> da respetiva estrutura assume um valor distinto (no caso em específico desta solução, foi-lhe atribuído o valor `queryID`), enquanto que o <attribute_value> toma o valor da respetiva declaração da query.

Adicionalmente, nesta `SingleExpression` são também armazenadas duas *flags*: a primeira indicando se o nó de query está associado a uma query que foi utilizada pelo utilizador e outra indicando se está associado a uma query executada no âmbito da inicialização da base de dados. O motivo da utilização destas *flags* prende-se com o fato de poderem ser repetidas queries por parte do utilizador. Assim sendo, caso (1) o utilizador submeta uma query que tenha sido utilizada no âmbito da preenchimento da malha inicial, apenas é necessário ativar a primeira flag definida. Por outro lado, (2) se o utilizador submeter queries repetidas, ao verificar que a query já foi executada pelo utilizador anteriormente, não é necessário fazer nada. Assim, ao serem executadas queries repetidas, as mesmas não são executadas, evitando assim também a replicação informação.

Preenchimento da Malha Inicial

No âmbito do preenchimento da malha inicial, são executadas queries a inquirir toda a informação das tabelas existentes no esquema da base de dados que se pretende proteger. Esta operação é descrita no algoritmo 4.

```
1 for table in database schema do
2 |   statement ← "select * from "+ table +";";
3 |   parse statement through QueryInterpreter;
4 |   execute query against original database and store its results through
  |    QueryExecutor;
5 end
```

Algoritmo 4: Preenchimento da Malha Inicial

Esta malha é apenas inicializada uma vez podendo, caso o utilizador pretenda, para cada caso de uso submetido ser reutilizada.

Execução de Queries do utilizador

A execução de queries submetidas pelo utilizador não recorre à base de dados original. Ao invés disso, recorre-se à malha inicial para obter os tuplos da mesma. Esta estratégia dispensa assim a utilização de técnicas de reescrita de queries, como era necessário no âmbito da solução utilizando grafos, apresentada na seção `Execução de queries - solução baseada`

em grafos do Anexo B. Ao invés disso, apenas necessita de trabalhar sobre as alterações efetuadas sobre a estrutura Query, no âmbito da transformação da cláusula select da query descrita em 4.4.3.

A seção Algoritmo de execução de queries submetidas por um utilizador do Anexo E resume esta operação.

Esta abordagem é mais rápida, no âmbito da execução e armazenamento de uma query submetida pelo utilizador, devido a

- não ter que comunicar com a base de dados, que se não estiver implementada na mesma máquina introduz um overhead significativo e
- não ter que, para cada valor de cada atributo, procurar no grafo pela referência do respetivo nó.

4.6 Gestor de Políticas

O Gestor de Políticas é responsável pela gestão das políticas de controlo de acesso probabilísticas e da flexibilidade dos atributos a usar no processo de avaliação. Para tal, recorre a um servidor de políticas, descrito na seção 4.6.1.

Este componente é utilizado em três fases da execução desta aplicação; primeiramente é através dele que se pode definir e armazenar as políticas de controlo de acesso probabilísticas que se pretendem implementar na fase da inicialização da aplicação. O segundo momento em que é utilizado corresponde à fase da interpretação de queries, tal como definido na subseção 4.4.3. Por fim, no âmbito da avaliação é necessário aceder às políticas de controlo de acesso probabilísticas definidas.

4.6.1 Desenvolvimento do Policy Server

No âmbito do desenvolvimento de um componente que pudesse armazenar as políticas de controlo de acesso supramencionadas, optou-se por apenas utilizar as coleções fornecidas pelo Java. Esta escolha justifica-se pelo fato de a informação necessária de armazenar ser relativamente reduzida. Posto isto, este servidor é meramente composto por um HashSet[1], onde são armazenados os atributos que não devem ser processados na fase de avaliação, e por um HashMap[2], onde são definidas as políticas de controlo de acesso probabilísticas: cada política de controlo de acesso (definida sob a forma de Query) é associada ao respetivo limite da probabilidade de acerto num tuplo sensível protegido.

4.7 Desenvolvimento do Evaluator

Nesta seção será descrito o processo de avaliação de informação passível de ser inferida por um utilizador *malicioso*. Convém mencionar que o processo descrito ao longo desta seção é repetido para cada tuplo sensível de cada política de controlo de acesso probabilística estabelecida. Adicionalmente, no âmbito de uma perspetiva mais pessimista do modelo de

[1] https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html
[2] https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

controlo de acesso implementado, o administrador pode definir momentos adicionais em que pretende efetuar o processo de avaliação, tal como ficou patente no capítulo 3.

Perante os exemplos que se seguem, convém explicar a *definição* da expressão `contexto corrente`: contexto corrente é formado pelo conjunto de valores dos atributos que compõem um tuplo sensível e dos atributos percorridos até um dado momento no âmbito de um caminho, juntamente com as cláusulas where das queries também percorridas.

O processamento realizado no âmbito do processo de avaliação é apresentado no algoritmo 5. É efetuado recursivamente para cada atributo sensível de predicado que se pretende obter. Na primeira chamada parte-se do contexto fornecido pelo tuplo sensível. As chamadas recursivas seguintes são efetuadas a partir de cada especificação do contexto corrente atual. Tal como se poderá perceber na subseção `Inferência de informação baseada em grafos`4.7.3, a forma como a informação inferida é associada ao tuplo sensível é representada na figura 4.4. Posto isto, para cada chamada recursiva subsequente, esta árvore continua a crescer a partir de cada uma das suas folhas.

Data: currentContext
Result: criticalAttributesValues

1 *resultCollections* ← `getCriticalAttributesValuesFromPaths(`*paths, currentContext*`)`;
2 initialize *collectionProcessed*;
3 **foreach** *collection in resultCollections* **do**
4 | filter critical attributes values from *collection* based on where clauses;
5 | *collectionProcessed* ← `aggregateTwoCollections(`*collectionProccessed, collection, currentContext*`)`;
6 **end**
7 remove overlapped tuples from *collectionProcessed*;
8 remove known invalid tuples from *collectionProcessed*;
9 normalize *collectionProcessed*;

Algoritmo 5: Obtenção dos valores dos atributos sensíveis de predicado

Este processamento começa com obtenção dos valores dos atributos sensíveis de predicado através de cada caminho passível de ser percorrido. Posteriormente, a cada conjunto de resultados obtidos (armazenados em coleções) são filtrados tuplos inválidos obtidos no âmbito de cada caminho, recorrendo a informação das cláusulas where, tal como descrito em 4.7.4. De seguida, utilizando os resultados filtrados anteriormente, agregam-se os resultados obtidos através dos múltiplos caminhos. Posteriormente, removem-se ocorrências repetidas de tuplos presentes na coleção processada, de acordo com a subseção 4.7.6 e os tuplos inválidos no âmbito do contexto corrente que o utilizador associa a outros contextos; tal como é apresentado na subseção 4.7.7. Por fim, é necessário normalizar a coleção com os resultados obtidos processados, tal como é descrito na subseção 4.7.8.

Posto isto, começar-se-á por explicar a forma como se obtêm os tuplos sensíveis em 4.7.1 e os caminhos entre os nós desses tuplos e os nós dos atributos sensíveis de predicado em 4.7.2.

Posteriormente será descrito como esta solução atravessa o grafo no âmbito de cada caminho em 4.7.3. No âmbito do atravessamento de um caminho poder-se-á perceber que em

alguns casos a quantidade informação com que é necessário lidar pode ser elevada. Neste contexto foi necessário desenvolver uma solução que permitisse acautelar estas situações. Posto isto, no âmbito da subseção 4.7.3 será descrita a solução utilizada no armazenamento da informação obtida no processamento de um caminho.

De seguida será explicada a forma como é inferida informação a partir das cláusulas where das queries submetidas pelos utilizadores, como se processam os resultados obtidos através de múltiplos caminhos em 4.7.5, 4.7.6, 4.7.7, 4.7.8 e a avaliação desses resultados em 4.7.11.

Por fim, será descrito o formato do relatório produzido no âmbito do processo de avaliação em 4.7.10.

4.7.1 Obtenção dos Tuplos Sensíveis

O processo de avaliação começa com a obtenção dos tuplos sensíveis. Apenas após a obtenção dos mesmos se tenta perceber se através de técnicas de inferência é possível descobrir os valores dos atributos sensíveis de predicado aos quais os tuplos estão associados. Um tuplo sensível apenas é considerado quando todos os valores dos atributos do mesmo tiverem sido obtidos. Deste modo, o administrador, aquando da definição das políticas de acesso, deve ter atenção aos atributos que adiciona à cláusula select de q_{ill}.

No âmbito da obtenção dos tuplos sensíveis foram utilizadas duas abordagens, apresentadas em 4.7.1 e 4.7.1. A diferença entre estas prende-se com a utilização da malha inicial por parte da segunda abordagem.

Solução baseada em técnicas de inferência

A obtenção dos possíveis tuplos sensíveis, apesar de à primeira vista parecer uma tarefa trivial, deve ser efetuada da forma como a busca de valores dos atributos sensíveis de predicado é tratada. Isto deve-se ao fato de o utilizador poder obter os respetivos tuplos sensíveis através de múltiplas queries e com uso a técnicas de inferência, sendo que neste caso o processo apresenta algumas nuances.

O processo de obtenção dos possíveis tuplos sensíveis começa com a pesquisa no grafo de todos os valores associados ao *primeiro* atributo que corresponde à chave do tuplo sensível. Após a obtenção desse valor, todos os restantes são obtidos recorrendo às estratégias de inferência apresentadas nesta seção. Neste caso, apenas se considera um tuplo quando se obtém valores efetivamente associados a ele. Exemplificando, suponha-se que o contexto da base de dados é apresentado na tabela 3.4 e que é definida a política de acesso q_{ill} e são executadas as queries apresentadas na listagem 4.4. Posto isto, pode-se constatar para os tuplos cujo (FirstName='Nancy') não é possível perceber qual dos tuplos retornados por q_1 estava associado a cada um dos tuplos { (EmployeeID=1), (FirstName='Nancy')}, { (EmployeeID=4), (FirstName='Nancy')} e { (EmployeeID=9), (FirstName='Nancy')}. Neste caso os respetivos tuplos não são considerados.

```
1 QILL: select EmployeeID, LastName, FirstName from Employees where Salary >=
    2500.00;
2
3 Q1: select LastName, FirstName, Country from Employees;
4 Q2: select EmployeeID, FirstName from Employees;
```

Listagem 4.4: Definição de caso de uso no âmbito da obtenção de tuplos sensíveis

Solução suportada pela malha inicial

A segunda solução, tal como já foi mencionado, destaca-se da primeira pela utilização da malha inicial. Foi desenvolvida no âmbito da solução final, utilizando a base de dados de hipergrafos. No processo de avaliação, esta malha é utilizada para validar os resultados obtidos a cada passo: ao atravessar um caminho, ao processar coleções resultantes de caminhos distintos, ente outros cenários.

No entanto, no âmbito da obtenção de tuplos sensíveis, por motivos de simplificação tanto em termos de código desenvolvido como de trabalho efetuado, optou-se por obter estes tuplos diretamente a partir desta malha.

Primeiramente, à semelhança do algoritmo 7, apresentado na seção `Algoritmo de execução de queries submetidas por um utilizador` do Anexo `E`, é necessário obter os tuplos completos das tabelas inquiridas por q_{ill}. De seguida, para cada tuplo, verifica-se se o utilizador inquiriu todos os valores dos atributos que compõem o tuplo sensível. Esta verificação é realizada através de cada query em conjunto com a malha inicial. Esta verificação é efetuada através do método `userAccessedThisTuple`.

Antes de descrever como funciona este método, é preciso de perceber como funciona o método `getAdminCount`. É através deste último que esta solução efetua a validação dos resultados obtidos, tal como se poderá perceber nas próximas seções.

A função do método `getAdminCount` corresponde à contagem do número de tuplos existentes na base de dados que são válidos (1) no contexto de alguns valores de atributos já fornecidos, tendo também em conta algumas (2) restrições adicionais. Esta contagem é efetuada sobre as tabelas que incluem (3) um conjunto de nomes de atributos.

Apresentando um exemplo para mais facilmente perceber o modo de funcionamento deste método, considere-se o esquema da base de dados apresentado em A.1 e suponha-se que se pretende obter a contagem de tuplos

- que possuem { (EmployeeID=4), (FirstName='Nancy')} - (1);
- válidos no contexto dado pela restrição (OrderID <= 25) - (2);
- pertencentes às tabelas que incluem os atributos { EmployeeID, FirstName, OrderID, UnitPrice} - (3).

Primeiramente, é necessário saber quais as tabelas em que se pretende obter a contagem de tuplos válidos (que obedecem a (1) e (2)). Para tal, através da instância da estrutura `DatabaseSchema`, pode-se perceber que para este esquema da base de dados o conjunto de tabelas que abrange o conjunto de atributos apresentado em (3) corresponde a { Employees, Orders, OrderDetails}. Posto isto, para cada tuplo completo das tabelas inquiridas (que respeite os valores fornecidos em (1)) verifica-se se o mesmo é válido no contexto de (2). Caso o seja, incrementa-se o contador.

Por fim, tendo em conta a a forma de funcionamento do método `userAccessedThisTuple`, apresentada na seção `Algoritmo do método userAccessedThisTuple` do Anexo `E`, pode-se então perceber que é possível a solução desenvolvida considerar um tuplo sensível mesmo quando o utilizador não efetuou queries que lhe permitam associar a informação. Um exemplo disto é através do conjunto de queries apresentadas na listagem 4.4, onde são inquiridos todos os valores dos atributos que compõem o tuplo sensível, sem, no entanto, ser possível associá-los, tal como mencionado na subseção anterior.

4.7.2 Determinação de Caminhos

O processo de determinação de caminhos simboliza a obtenção das possíveis formas de um utilizador ter conseguido associar valores do tuplo sensível a valores de atributos sensíveis de predicado. Esta associação pode ser feita através de múltiplas projeções de atributos entre as queries que formam cada caminho.

O processo que permite a descoberta das possíveis sequências de passos para a obtenção dos possíveis valores dos atributos foi designado de `Determinação de Caminhos`. Exemplificando, para o caso apresentado na figura 4.3, se se pretender obter um caminho entre valores do atributo `CustomerID` e valores de `Country`, é possível um utilizador associar valores destes atributos através de um caminho de três passos: no primeiro relaciona `CustomerID` com `City`, o segundo corresponde a um passo de `transição de query`, e o terceiro associa valores de `City` a `Country`.

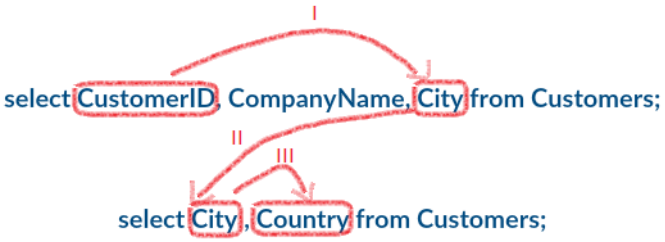

Figura 4.3: Exemplo de Caminho Simples

Na seção `Algoritmo de Determinação de Caminhos` do Anexo `E` é apresentado o processo base da determinação de caminhos.

Após a obtenção todos os caminhos possíveis, filtra-se o resultado removendo os caminhos (1) que não contenham no primeiro passo pelo menos um atributo pertencente ao tuplo sensível e os (2) que corresponderem a *super-caminhos*[3] de outros nesse mesmo conjunto.

A filtragem efetuada em (1) é necessária uma vez que o ponto de partida corresponderá sempre aos nós que contêm os valores dos atributos sensíveis. Por outro lado, a filtragem efetuada em (2) serve para evitar ter que percorrer o mesmo caminho vezes repetidas.

Por fim, junta-se ao conjunto de caminhos que já estão marcados para avaliação (Conj_1) um conjunto de caminhos adicional (Conj_2). Este conjunto adicional é formado por caminhos formados apenas por uma query que inquire os atributos sensíveis de predicado que se pretende encontrar. Para formar o conjunto (Conj_2) utilizam-se apenas as queries que não foram utilizadas no âmbito de (Conj_1).

Filtragem adicional recorrendo à malha inicial

Adicionalmente, utilizando o método já apresentado `getAdminCount` (que recorre à malha inicial), filtram-se os caminhos dos quais se sabe à partida que não resultará informação associada ao respetivo tuplo sensível. Esta filtragem adicional tem como objetivo evitar processamento desnecessário.

[3]Um caminho c_1 é *super-caminho* de um caminho c_2 caso $c_2 \subset c_1$, ou seja, todos os passos de c_2 estiverem contidos em c_1

Relembrando, o método `getAdminCount` necessita, para obter o número de tuplos válidos: o contexto (1) de valores de atributos já definidos, (2) restrições adicionais e (3) o conjunto de nomes de atributos que ditarão as tabelas que deverão ser atravessadas. Posto isto, para verificar se para um dado tuplo sensível um caminho retorna informação válida no contexto do mesmo, atribui-se a (1) os valores dos atributos que compõem o tuplo sensível, a (2) as restrições das queries que compõem um dado caminho e a (3) todos os nomes de atributos inquiridos e presentes nas cláusulas where das queries que formam o caminho. Se a contagem for igual a zero remove-se o caminho dos que serão utilizados no processo de avaliação.

De fato, tal como se poderá perceber no âmbito da subseção `Processamento de múltiplos caminhos recorrendo a resultados inválidos` 4.7.5, apenas se filtram os caminhos cujo contador de tuplos válidos até ao penúltimo passo seja zero.

4.7.3 Inferência de informação baseada em grafos

Tal como mencionado em `Travessia do Grafo` 3.3.1, o processo de atravessamento do grafo simboliza o processo de inferência de informação que um utilizador pode efetuar.

Foram também mencionados os princípios aos quais é necessário obedecer quando se pretende atravessar o grafo:

1. avaliação da cláusula where da query que compõe cada passo no caminho, no âmbito do contexto corrente;

2. filtragem dos tuplos obtidos da query que compõe cada passo no caminho.

O processamento requerido em `(1)`, pode ser resolvido da mesma forma como se avalia se um tuplo sensível é protegido ou não, ou seja, recorrendo a recursividade. Caso o resultado dessa avaliação seja válido, pode-se avançar; caso contrário (o utilizador não ter informação suficiente para validar a cláusula where ou saber que a cláusula é inválida no âmbito do contexto corrente) interrompe-se o atravessamento do caminho.

Posto isto, pode-se perceber que a triagem de caminhos efetuada no âmbito da subseção 4.7.2, corresponde apenas a um filtro inicial, uma vez que a validação das cláusulas where das queries presentes no caminho estarão dependentes da informação a que o utilizador efetivamente teve acesso.

O processamento requerido em `(2)`, é dividido em duas fases.

Na primeira divide-se o conjunto dos atributos inquiridos pela query em dois. Um conjunto, C_{all_rel}, com os atributos cujos todos valores obtidos estarão associados ao contexto corrente e outro conjunto, $C_{not\_all\_rel}$, com os atributos cujos valores poderão não estar todos associados ao contexto corrente.

O exemplo apresentado no âmbito da listagem 3.6, para o caminho C = {q_2, q_4} pode-se perceber que no primeiro passo do caminho, a presença do atributo CustomerID na cláusula select de q_2 indica que os tuplos através dela obtidos estarão efetivamente associados ao contexto atual. Neste caso, o C_{all_rel} contém todos os atributos inquiridos por q_2, enquanto que $C_{not\_all\_rel}$ fica vazio. Por outro lado, no último passo, o fato de a projeção de atributos ser feita sobre o atributo EmployeeID, não garante que os resultados obtidos a partir de q_4 estejam efetivamente todos associados ao contexto corrente. Ou seja, C_{all_rel} = { EmployeeID} e $C_{not\_all\_rel}$ = { OrderDate, UnitPrice}.

Esta solução distingue estes dois tipos de resultados de acordo com o esquema dos atributos que (1) compõem o contexto corrente e (2) inquiridos pela query do passo que se pretende atravessar.

O algoritmo 10 definido na seção `Algoritmo de determinação de tipos de resultados de um atributo inquirido` do Anexo `E` apresenta a forma como é determinado se os resultados obtidos para esse mesmo atributo são todos associados ao contexto corrente (ALL_RELATED), ou não (NOT_ALL_RELATED).

Considere-se um caso em que o contexto corrente contém os valores dos atributos { Customers.CustomerID, Customers.ContactName, Customers.CompanyName, Orders.OrderID, Employees.EmployeeID} e que a query que se pretende atravessar inquire os atributos { Customers.CustomerID, Employees.EmployeeID, Customers.Country, Orders.OrderDate}. Neste caso, os valores do atributo Customers.Country obtidos estarão todos associados ao contexto corrente visto que é inquirido o atributo Customers.CustomerID que compõe a chave da tabela Customers, que corresponde à tabela original do atributo Customers.Country. No entanto, para o atributo Orders.OrderDate (cuja tabela *original* corresponde a Orders) tal não se verifica. Apesar de serem inquiridas as chaves das tabelas Customers e Employees, o contexto corrente possui atributos da tabela Orders, sendo que ela pertence ao caminho entre Employees→Orders (composto pelas tabelas {Employees, Orders}) e Customers→Orders (composto pelas tabelas {Customers, Orders}).

Após dividir os atributos nos dois conjuntos supramencionados, é necesário obter os valores dos atributos pertencentes de $C_{not\_all\_rel}$. Tal como mencionado, também, em `Travessia do Grafo` 3.3.1, este processamento torna-se necessário pois, através da informação obtida resultante da execução de outras queries, é possível que um utilizador consiga inferir mais informação relativa aos valores dos atributos do que aquela que esta query oferece.

À semelhança da validação das cláusulas where, a filtragem dos resultados obtidos recorre também a recursividade. De fato, pode-se olhar para o processamento necessário para filtrar os resultados obtidos numa dada query como uma parcela do processamento necessário para efetuar a avaliação de uma cláusula where. Para avaliar uma cláusula where é necessário:

1. obter os valores dos atributos sensível de predicado (através das várias técnicas de inferência de informação);

2. processar os resultados obtidos através dessas técnicas;

3. avaliar as restrições da cláusula where no contexto dos resultados obtidos processados.

Ou seja, se para efetuar a validação das cláusulas where é necessário realizar estes três passos, a filtragem dos resultados obtidos corresponde *apenas* à realização dos dois primeiros passos.

Além do apresentado na subseção 3.3.1, outro exemplo da necessidade de aplicação deste processamento corresponde ao apresentado na listagem 4.5.

```
1 QILL: select CustomerID, CompanyName, ContactName from Customers join Orders on
      Customers.CustomerID = Orders.CustomerID join OrderDetails on Orders.
    OrderID = OrderDetails.OrderID and UnitPrice > 50.00;
2
```

```
Q1: select OrderID, OrderDate, Customers.ContactName from Orders join Customers
    on Orders.CustomerID = Customer.CustomerID;
Q2: select OrderID, CustomerID, EmployeeID from Orders where ShippedDate > '
    1996-07-15';
Q3: select CustomerID, CompanyName, ContactName from Customers;
Q4: select EmployeeID, OrderDate, UnitPrice from Orders join OrderDetails on
    Orders.OrderID = OrderDetails.OrderID;
```

Listagem 4.5: Definição de caso de uso no âmbito da técnica de inferência de informação baseada em grafos

Partindo dos valores dos atributos que compõem um tuplo sensível, para o caso do caminho C = { q_1, q_4}, pode-se constatar que em ambos os passos, a informação obtida não está toda associada ao contexto corrente.

No entanto, prestando especial atenção ao primeiro passo, pode-se perceber que *em paralelo* com q_1 é possível, através de q_2 que o utilizador consiga perceber alguns valores de OrderID como estando efetivamente associados ao tuplo sensível em questão; dado que a projeção entre os atributos inquiridos em q_2 e os atributos que compõem o contexto corrente é efetuada sobre o atributo CustomerID; podendo através do processo explicado no algoritmo 10 inferir que a informação obtida no âmbito de q_2 está efetivamente toda associada ao contexto corrente.

Esta informação que um utilizador vai obtendo à medida que percorre um caminho pode ser representada sob a forma de uma árvore n-ária, onde à medida que é atravessada em profundidade (atravessando um ramo e partindo para os seus respetivos filhos) o contexto corrente vai sendo cada vez mais especificado. Atendendo ao exemplo anterior, para o caminho C = {q_1, q_4} pode-se perceber que a informação associada a cada tuplo sensível vai aumentando de acordo com a representação apresentada na figura 4.4.

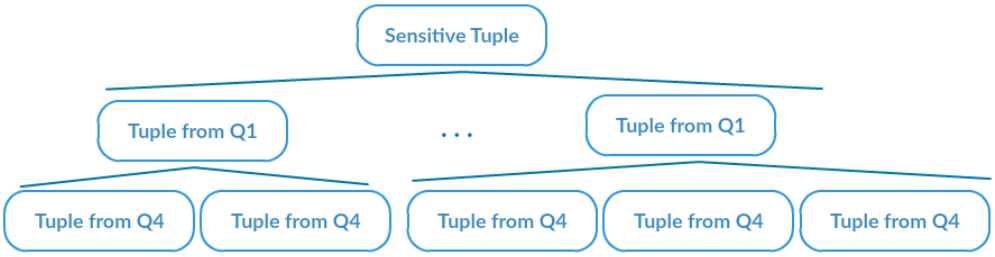

Figura 4.4: Representação da informação adquirida ao longo de um Caminho

Para armazenar os resultados obtidos foi necessário desenvolver coleções suportadas em disco. O recurso a este paradigma prende-se com a quantidade de informação necessária processar no âmbito do processo de avaliação.

Como se pôde perceber até este momento, as coleções necessitarão de armazenar a informação de acordo com a representação efetuada na figura 4.4.

Começando pelo tipo de informação que é obtida no âmbito de cada query num dado caminho, foi necessário definir cada coleção adequadamente:

- `ALL_RELATED` para o caso de se saber que a informação está toda associada ao contexto corrente;

- `NOT_ALL_RELATED` para o caso de não se saber se a informação está toda associada ao contexto corrente.

De fato, no exemplo baseado na listagem 4.5 pôde-se perceber que mesmo num dado passo é possível que os resultados obtidos sejam dos dois tipos: ALL_RELATED e NOT_ALL_RELATED.

A figura 4.5 representa então a distinção do tipo de coleções/ informação obtida ao longo do caminho C = { q_1, q_4}, onde a notação REL simboliza o tipo ALL_RELATED e a notação NAR simboliza NOT_ALL_RELATED.

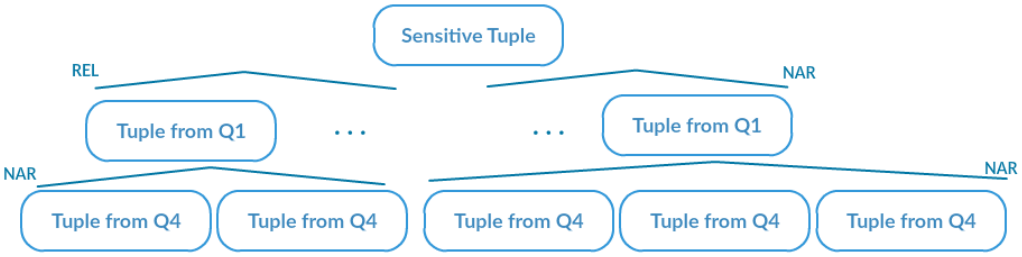

Figura 4.5: Utilização de coleções para armazenar a informação adquirida ao longo de um Caminho

Em termos de organização da informação optou-se por, sempre que os resultados obtidos no âmbito de uma query sejam ALL_RELATED para um ramo R1 já existente, não criar uma coleção filha para esse ramo. Ao invés disso, ao nível da coleção de R1, substituiu-se R1 por um conjunto de ramos sendo cada um composto por um tuplo retornado pela query anexado à informação já existente em R1. A explicação para esta opção prende-se unicamente com fato de esta abordagem se tornar mais *natural* na medida em que a informação fica acondicionada apenas nas coleções necessárias. No limite, após atravessar um caminho existe apenas uma coleção onde está inserida toda a informação em vez de uma árvore de profundidade igual ao número de passos do respetivo caminho.

No âmbito do desenvolvimento das coleções foram utilizadas duas abordagens distintas. A primeira, suportada em ficheiros, foi desenvolvida para a primeira versão do trabalho efetuado nesta dissertação, baseada em grafos. Devido a isto, é descrita na seção `Desenvolvimento de coleções suportadas por ficheiros` do Anexo `B`. Relativamente à segunda abordagem, suportada por uma base de dados, correspondendo à solução adotada no âmbito da solução final, é descrita na subseção `Desenvolvimento de coleções suportadas por base de dados de hipergrafos` 4.7.3.

Desenvolvimento de coleções suportadas por base de dados de hipergrafos

As coleções suportadas por hipergrafos foram desenvolvidas sobre a mesma instância da base de dados HyperGraphDB, utilizada no âmbito do trabalho desenvolvido. O princípio usado para desenvolver estas coleções corresponde ao mesmo utilizado para armazenar a informação obtida por cada query: utiliza-se um nó de coleção como *ponto de ancoragem*, em que cada aresta (ao qual o nó de coleção deve estar associado) corresponde a um ramo (especificação) do contexto corrente.

De fato, através da forma como a camada responsável por modelar a estrutura do hipergrafo está organizada, apresentada na seção 2.2.3, esta ferramenta permite que uma aresta possa conter outras arestas. Isto permite que no âmbito de cada linha da coleção seja possível armazenar diretamente os conjuntos de tuplos que formam o ramo do contexto corrente, em vez de, à semelhança da solução suportada em ficheiro, necessitar de armazenar cada valor individualmente.

A título de exemplo, a representação de uma linha de uma coleção após a execução das queries q_1 e q_2 é apresentada na figura 4.6.

```
1 Q1: select OrderID, EmployeeID, CustomerID from Orders;
2 Q2: select OrderID, OrderDate, RequiredDate, EmployeeID from Orders;
```

Listagem 4.6: Definição de caso de uso no âmbito da criação de coleções

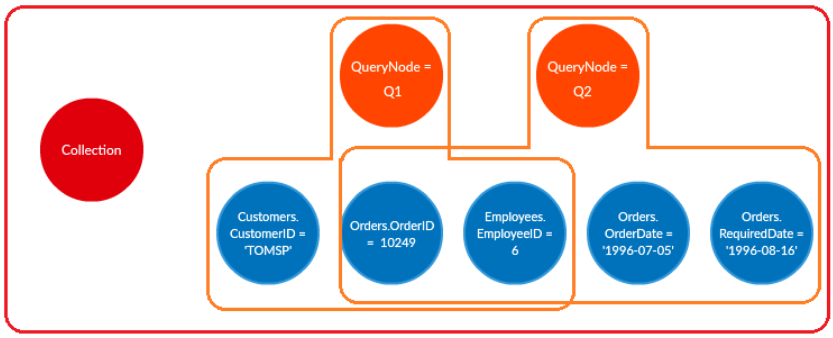

Figura 4.6: Representação de coleção suportada por HyperGraphDB

Deste modo, pode-se então perceber que quando se pretende atravessar as coleções *em profundidade*, basta alterar o nó de coleção sobre o qual se obtêm as próximas especificações do contexto corrente. Tendo em conta a forma como as coleções estão ligadas, onde a coleção filha aparece num ramo (especificação) do contexto corrente da coleção raiz, que é simbolizado por uma aresta incidente nos dois respetivos nós de coleção, é necessário ter um mecanismo que previna que quando se quer atravessar a coleção filha (obter as especificações do contexto em que a mesma surge) *se volte à coleção mãe*.

Posto isto, à semelhança das coleções suportadas por ficheiros, recorreu-se ao método `System.nanoTime()` para obter o identificador de cada coleção. No entanto foi necessário alterar a nomenclatura utilizada no âmbito das coleções suportadas por ficheiros para:

<collection_identifier>_<result_type>_<root_collection_identifier>

onde os componentes <collection_identifier> e <result_type> se mantiveram, surgindo <root_collection_identifier> que contém ao identificador da coleção raiz (no caso de possuir). Caso não possua uma coleção raiz, o campo <root_collection_ identifier> permanece em branco. Através desta nomenclatura, quando se está a iterar sobre todas as arestas do nó de uma coleção $Coll_1$, se se verificar que no âmbito de uma aresta e_1 se obtém um nó

de coleção cujo <collection_identifier> seja igual ao <root_collection_ identifier> de $Coll_1$, descarta-se e_1.

Acrescentando a query q_3 às queries q_1 e q_2 do exemplo anterior, a representação de uma linha das respetivas coleções criadas é apresentada na figura 4.7. Pode-se então constatar que a aresta e_1 abrange os dois nós de coleção e também os tuplos retornados por q_1 e q_2, enquanto que a aresta e_2 abrange o nó da coleção filha juntamente com o tuplo retornado por q_3.

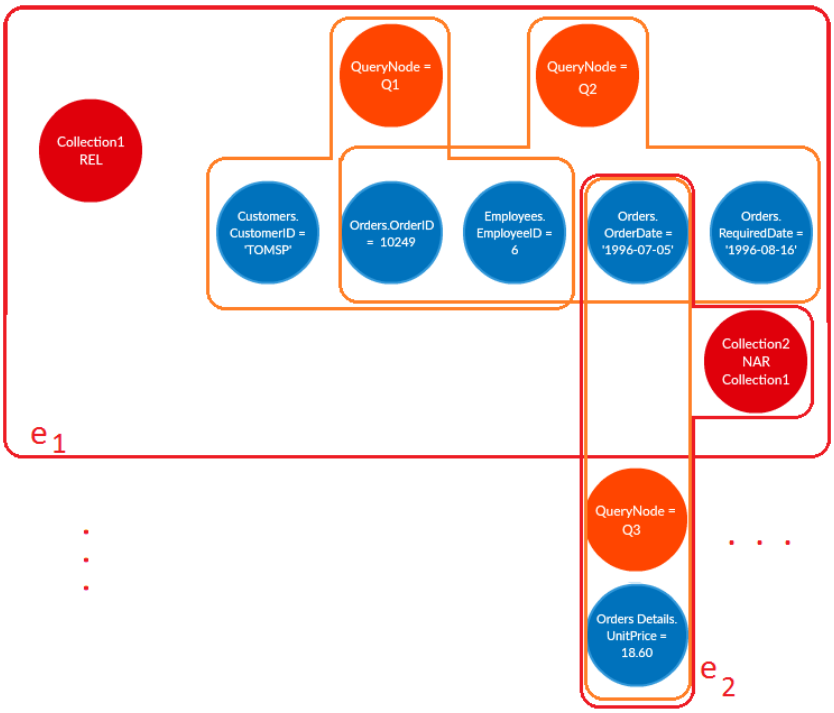

Figura 4.7: Representação do encadamento de coleções suportadas por HyperGraphDB

Evolução do contexto corrente e utilização da malha inicial

Tal como já foi referido, esta solução recorre à malha inicial para ir validando a informação que obtém durante o processo de avaliação. No âmbito do atravessamento de um caminho, esta utilização destina-se à verificação da validade dos conjuntos de tuplos retornados por cada query que o compõem. Esta validação é efetuada através do método `getAdminCount` sendo que o resultado da mesma é positivo caso o contador de tuplos válidos seja maior do que zero e negativo caso contrário. É então realizada quando

- se adiciona um tuplo que se sabe estar associado ao contexto corrente (ou seja, que corresponde a um resultado do tipo REL);
- se adiciona uma coleção de resultados que à partida não estão todos associados ao contexto corrente (que pertencem ao tipo de resultados NAR).

Naturalmente, apenas se adicionam estes resultados caso a validação efetuada tenha um resultado positivo. Este resultado é então positivo quando, para o primeiro caso o tuplo está

efetivamente associado ao contexto corrente e, para o segundo caso, quando existe pelo menos um tuplo dos que foi adicionados à coleção NAR que está associado ao contexto corrente.

Em relação ao primeiro caso é fácil perceber a sua utilidade. No entanto, em relação ao segundo caso, a utilização desta validação tem uma consequência que pode não ser evidente à primeira vista. Recorrendo ao exemplo apresentado na listagem 4.5, constatou-se que no âmbito do primeiro passo era possível, para os casos das Orders cujo ShippedDate > '1996-07-15', obter tuplos de q_2 que o utilizador sabia estarem efetivamente associados ao contexto corrente. Posto isto, parte dos resultados obtidos nesse passo seriam REL enquanto que os restantes pertenceriam ao tipo NAR. A figura 4.5 ilustrava esta divisão.

Neste contexto, suponha-se agora que um dado Customer estava associado apenas a Orders cujo ShippedDate > '1996-07-15'. Neste caso, através da informação inferida de q_2, era possível obter todos os tuplos válidos de q_1 no contexto desse Customer em particular, como pertencendo ao conjunto de resultados obtidos REL. Adicionalmente, podía-se obter outros tuplos de q_1 que seriam inseridos no conjunto de resultados do tipo NAR. Nesta situação o conjunto de resultados NAR não conteria nenhum resultado válido no contexto do Customer supramencionado.

Pode-se então perceber que o resultado da validação efetuada sobre o conjunto NAR recorrendo à malha inicial seria negativo. Mesmo não tendo sido submetidas queries suficientes para que um utilizador percebesse que esse Customer estava efetivamente associado apenas a Orders com ShippedDate > '1996-07-15', esta solução, ao descartar o conjunto de resultados NAR, partia do princípio que o utilizador tinha tido acesso a informação suficiente para poder inferir esta situação.

A utilização do método `getAdminCount` para validar os resultados obtidos a cada passo, sendo bastante importante no âmbito desta solução, de acordo com o que foi descrito até agora, possui uma grande limitação.

Recorrendo ao exemplo apresentado na listagem 4.5, para o caminho caminho C = { q_1, q_4} percebe-se que no primeiro passo, a informação obtida não estava toda associada ao contexto corrente. Neste caso, para um tuplo retornado por q_1 que efetivamente não está associado ao contexto de um dado Customer, no âmbito da execução de q_4, a validação de todos os tuplos retornados por esta query terá sempre um resultado negativo. Como consequência, pode-se então perceber que exceto para o último passo de um caminho, os resultados obtidos estariam todos associados ao contexto de base.

Por se entender que o resultado de uma solução assim desvirtuaria a motivação inicial desta dissertação, foi necessário desenvolver uma estratégia para evitar que este cenário acontecesse.

A abordagem para a solução deste problema passou por fazer o contexto utilizado para validar os resultados obtidos em cada passo *evoluir* da seguinte forma:

- quando a coleção em que o contexto corrente é especificado é do tipo NAR, são utilizados como pressupostos para a validação dos resultados obtidos no próximo passo do caminho apenas os valores dos atributos que compõem essa coleção bem como as restrições das respetivas queries;

- quando a coleção em que o contexto corrente é especificado é do tipo REL, são utilizados como pressupostos para a validação dos resultados obtidos no próximo passo do caminho os pressupostos utilizados anteriormente, juntamente com os valores dos atributos que compõem essa coleção e as restrições das respetivas queries.

Relembrando, na invocação o método `getAdminCount` é necessário fornecer (1) o contexto dos valores de atributos, (2) as restrições em que esses valores foram obtidos e (3) um conjunto de nomes de atributos que servirão para determinar as tabelas sobre as quais se obterá a contagem de tuplos válidos que respeitem (1) e (2).

Posto isto, no âmbito do exemplo anterior, para um tuplo t_1 pertencente ao conjunto de resultados do tipo NAR obtido através de q_1, aquando da obtenção dos resultados de q_4, o método `getAdminCount` seria invocado com

- (1) - os valores dos atributos de t_1 e de cada tuplo obtido através de q_4;
- (2) - as restrições de q_1 e q_4, que neste caso não existem;
- (3) - os nomes dos atributos apresentados pertencentes a (1).

sendo posteriormente efetuada a validação de acordo com a descrição já efetuada nesta subseção.

4.7.4 Inferência de informação baseada em cláusulas where

Como referido na subseção `Inferência de informação baseada em cláusulas where` 3.3.1, as cláusulas where das queries submetidas, ao estabelecerem o contexto em que a informação por elas é obtida, podem ser utilizadas para

- filtrar valores de atributos obtidos através de técnicas de inferência baseadas em projeções de atributos comuns entre queries;
- avaliação *direta* do contexto corrente no âmbito do processo de avaliação de uma política de controlo de acesso estabelecida.

No âmbito desta estratégia, para associar o contexto corrente à restrição da cláusula where de uma query, recorre-se aos atributos inquiridos por ela: se o contexto estiver associado a valores retornados pela query em questão, então há a possibilidade de ser abrangido pela restrição.

Para obter os atributos inquiridos pela query em questão, à semelhança da filtragem dos tuplos obtidos numa query que compõe um passo num caminho, recorre-se a recursividade.

Associação válida entre o contexto corrente e tuplos retornados pela query

Perante o cenário de se conseguir associar o contexto corrente a um tuplo ou um conjunto de tuplos retornados pela query em questão, pode-se considerar que o mesmo é abrangido pela restrição da cláusula where se:

- (1) o utilizador obtém o tuplo e sabe que ele está efetivamente associado ao contexto corrente (ou seja, obtido através de passos do tipo REL), (2) o mesmo está associado inequivocamente à restrição da cláusula where e (3) o utilizador sabe-o;
- (1) o utilizador obtém um conjunto de tuplos onde existe pelo menos um que está efetivamente associado ao contexto corrente (ou seja, obtido através de passos do tipo NAR), (2) todos eles estão associados inequivocamente à restrição da cláusula where e (3) o utilizador sabe-o.

Pode-se depreender que um tuplo inquirido está associado inequivocamente à restrição da cláusula where e o utilizador sabe-o caso o tuplo for formado:

1. pelas chaves primárias de tabelas cujas relações com as tabelas dos atributos que estão presentes na cláusula where sejam do tipo 1:1 ou N:1, pela propriedade de unicidade das chaves primárias;
2. pelas chaves primárias de tabelas cujas relações com as tabelas dos atributos que estão presentes na cláusula where sejam do tipo 1:N e o contexto corrente não contenha atributos que pertencem a tabelas cuja relação com as tabelas das chaves primárias inquiridas seja do tipo 1:N, pela propriedade de unicidade das chaves primárias;
3. por atributos comuns que estão apenas associados a valores de atributos que respeitam a respetiva cláusula where e o utilizador sabe-o;
4. por atributos comuns que estão apenas associados ao contexto corrente e o utilizador sabe-o.

Exemplificando, considere-se o estado da base de dados apresentado na tabela 3.4 e que um utilizador tem permissão para executar as queries apresentadas na listagem 4.7

```
1 Q1: select EmployeeID, FirstName, LastName from Employees;
2 Q2: select FirstName, Country from Employees where Salary > 2000;
```

Listagem 4.7: Definição de caso de uso no âmbito de uma associação válida entre contexto corrente e cláusula where (1)

Posto isto, o utilizador:

- pode inferir que o tuplo sensível definido por (EmployeeID = 6) possui um (Salary > 2000), uma vez que o respetivo tuplo é retornado em q_2, existe apenas um empregado com (FirstName = 'Michael') e através de q_1 é possível que o utilizador adquira esse conhecimento.
- por exclusão de partes, pode inferir que o tuplo sensível definido por (EmployeeID = 7) possui um (Salary <= 2000), uma vez que apenas existe um empregado com (FirstName = 'Robert'), o utilizador através da informação retornada por q_1 sabe-o e esse nome não foi retornado por q_2.
- não pode inferir quais os tuplos sensíveis com (FirstName = 'Nancy') uma vez que existe mais do que um empregado que possui este nome, e não é possível saber a qual se refere cada tuplo retornado por q_2. Por outro lado, se no estado da base de dados todos os empregados com (FirstName = 'Nancy') tivessem (Salary > 2000), ao serem retornados os três tuplos com (FirstName = 'Nancy') existentes na base de dados, poder-se-ia inferir que todos os tuplos sensíveis com (FirstName = 'Nancy') possuíam (Salary > 2000).

No âmbito do desenvolvimento desta técnica de inferência de informação, tal como já foi mencionado, recorreu-se a recursividade para obter os valores dos atributos que são inquiridos pela respetiva query.

A validação através da qual se verifica se cada tuplo obtido no passo anterior está associado inequivocamente à restrição da cláusula where e o utilizador o sabe, é efetuada recorrendo à malha inicial. Nesta situação recorre-se ao paradigma `soft computing`[4] e, tal como na obtenção dos tuplos sensíveis e validação de resultados obtidos a cada passo de um caminho, a utilização desta malha leva a que em alguns cenários a informação inferida pela solução desenvolvida possa exceder a informação passível de ser inferida por um utilizador, tal como se poderá perceber de seguida. Apesar disto, considera-se esta aproximação vantajosa na medida em que diminui a complexidade necessária deste processamento.

Ao recorrer a esta abordagem não é avaliado se os tuplos estão inequivocamente associados à restrição da cláusula where. Ao invés disso:

- para um tuplo t_1 efetivamente associado ao contexto corrente (obtido quer através de passos REL, quer através de passos NAR), é avaliado se o contexto formado pelo contexto corrente juntamente com o tuplo t_1 é válido no âmbito da restrição da cláusula where.

- para um tuplo t_2 que não está associado ao contexto corrente, é avaliado se o contexto formado por t_2 é válido no âmbito da restrição da cláusula where.

Ao recorrer a esta abordagem, não se tem em conta o fato de o utilizador saber se os valores dos atributos obtidos estão apenas associados a valores de atributos que respeitam a cláusula where da respetiva query ou se estão apenas associados ao contexto corrente.

Posto isto, considere-se o estado da base de dados apresentado na tabela 3.4 e o exemplo apresentado na listagem 4.8.

```
1 Q1: select EmployeeID, FirstName, LastName from Employees;
2 Q2: select FirstName, Country from Employees;
3 Q3: select Country from Employees where Salary >= 2000;
```

Listagem 4.8: Definição de caso de uso no âmbito de uma associação válida entre contexto corrente e cláusula where (2)

Neste caso, para o tuplo sensível identificado por EmployeeID = 4, a solução desenvolvida consegue perceber que o mesmo está associado à cláusula where de q_3, apesar de um utilizador através deste conjunto de queries não o conseguir inferir.

Através da query q_2 é possível saber que o respetivo empregado está associado a (Country = 'USA') ou (Country = 'UK'). Como este resultado é do tipo NAR, é necessário que todos os tuplos obtidos estejam associados à restrição da cláusula where de q_3.

Para (Country = 'UK'), recorrendo à malha inicial, ao se verificar que o mesmo não está associado ao contexto do empregado definido por EmployeeID = 4, verifica-se para o contexto formado por { (Country = 'UK')} se o mesmo é válido no âmbito da restrição da cláusula where de q_3. Como o resultado desta validação é positivo (visto que existe que o empregado identificado por EmployeeID = 6 cujo Country = 'UK' que possui um Salary = 2004.07 >= 2000), avança-se para o tuplo (Country = 'USA'). Neste caso, como o respetivo tuplo está associado ao empregado cujo EmployeeID = 4, verifica-se que o contexto formado por { (EmployeeID = 4), (LastName = Dodsworth) (FirstName = Nancy), (Country = 'USA')} também é válido. Posto isto, pode-se então concluir que este empregado possui (Salary >=

[4] https://en.wikipedia.org/wiki/Soft_computing

2000).

Por fim, é necessário considerar o caso de a cláusula where ser formada por um, ou mais, operadores lógicos `or`.

Se a cláusula where for definida por `(Employees.Salary >= 2000 or Employees.Country = 'USA')`, é possível a restrição ser validada caso se verifique uma das seguintes condições: (1) `(Employees.Salary >= 2000)`, (2) `(Employees.Country = 'USA')` ou (3) `(Employees.Salary >= 2000 and Employees.Country = 'USA')`. Posto isto, caso seja possível associar o contexto corrente a tuplos retornados pela query, apenas se pode inferir inequivocamente que um dado empregado possui (Employees.Salary >= 2000) se se souber que o mesmo não está associado a (Employees.Country = 'USA').

Perante este caso, à semelhança da validação das cláusulas where das queries na Travessia de um Caminho, recorreu-se a recursividade para avaliar se cada uma das restantes condições que formam a cláusula where são inválidas no âmbito do contexto corrente.

Associação inválida entre o contexto corrente e tuplos retornados pela query

Por outro lado, pode ocorrer o caso de não se conseguir associar o contexto corrente a nenhum tuplo retornado pela query. Perante este cenário, apenas se pode concluir que o contexto corrente está associado à restrição complementar da cláusula where da respetiva query se se verificar que todos os possíveis tuplos associados ao contexto corrente foram obtidos.

Considere-se o caso apresentado na listagem 4.9. Para o contexto corrente composto pelos valores dos atributos {CustomerID, CompanyName, ContactName}, considere-se o caso em que se pretende perceber se o mesmo está associado à cláusula where de q_4. Neste caso, suponha-se que todas as OrderID às quais o cliente é associado através de q_2 e q_3, não estão associadas a OrderDate >= '1997-05-01', logo não são devolvidas por q_4. Perante esta situação, um utilizador não pode inferir que o cliente não está associado a (OrderDate >= '1997-05-01'), visto que não sabe se obteve todas as compras associadas ao respetivo cliente.

```
1 Q12: select CustomerID, CompanyName, ContactName from Customers;
2 Q46: select CustomerID, OrderID from Orders where OrderID < 1500;
3 Q47: select CustomerID, OrderID from Orders where EmployeeID >= 5;
4 Q48: select OrderID from Orders where OrderDate >= '1997-05-01';
```

Listagem 4.9: Definição de caso de uso no âmbito de uma associação inválida entre contexto corrente e cláusula where

No entanto, perante este cenário, esta solução, recorrrendo à malha inicial consegue identificar se foram inquiridos todos os valores de OrderID associados ao contexto corrente ou não. Neste caso, pode-se constatar que é possível associar o CustomerID a OrderID através da utilização de dois caminhos: $C_1 = \{q_2\}$ e $C_2 = \{q_3\}$.

Correspondendo o contexto de um caminho à interseção das restrições das cláusulas where das queries que o compõem, tem-se então que o contexto de C_1 é dado por Ctx_1 = (OrderID < 1500) e o contexto de C_2 é dado por Ctx_2 = (EmployeeID >= 5).

Posto isto, recorrendo à malha inicial deteta-se se se obteve todos os OrderID associados a um dado Customer comparando os valores de contagens obtidas através do método `getAdminCount`. Sendo assim, começa-se por obter o número de valores ($count_1$) de OrderID

associadas ao contexto corrente composto pelos valores dos atributos {CustomerID, CompanyName, ContactName}. De seguida, obtém-se a contagem de Orders ($count_2$) associadas ao contexto corrente, obtidas através do contexto do caminho C_1; ou seja, a contagem de OrderID associadas ao Customer que possuem (OrderID $<$ 1500). Por fim, obtém-se a contagem de Orders ($count_3$) associadas ao contexto corrente, obtidas através do contexto do caminho C_2 e que não foram obtidas no contexto de C_1; ou seja, a contagem de OrderID associadas ao Customer que possuem (EmployeeID $>=$ 5 $\wedge$ OrderID $>=$ 1500).

Posto isto, para verificar se foram obtidas todos as OrderID associados a um dado Customer, verifica-se se ($count_1$ == $count_2$ + $count_3$). Esta condição ser verdadeira simboliza o fato de terem sido obtidas todas as OrderID associadas ao Customer.

Filtragem de valores obtidos através de um caminho

A filtragem dos conjuntos de valores dos atributos processa-se de acordo com o descrito ao longo desta subseção. No entanto é necessário realizar este processo para cada especificação do contexto corrente. Relembrando a forma como a informação inferida é associada ao tuplo sensível, representada na figura 4.4, isto é equivalente a dizer que é necessário efetuar este processo para cada folha desta árvore.

Pode-se perceber a necessidade deste processamento ser efetuado para cada especificação através do exemplo apresentado no âmbito do caso de uso definido na listagem 4.10.

```
1 Q1: select OrderID, EmployeeID, CustomerID from Orders;
2 Q2: select CustomerID, CompanyName, ContactName from Customers;
3 Q3: select EmployeeID, OrderDate, UnitPrice from Orders join OrderDetails on
    Orders.OrderID = OrderDetails.OrderID;
4 Q4: select CustomerID from Orders join OrderDetails on OrderDetails.`OrderID` =
     `Orders`.`OrderID` where UnitPrice > 50;
5 Q5: select OrderID from OrderDetails where UnitPrice > 45;
```

Listagem 4.10: Definição de caso de uso no âmbito da filtragem de valores obtidos recorrendo a cláusulas where

Perante este caso de uso, considere-se o caminho C_1 = { q_2, q_1, q_3} e que um dado CustomerID c_1 é retornado por q_4 e uma OrderID o_1 associada a c_1 não é retornada por q_5. Perante este cenário, considere-se também que o EmployeeID e_1, associado a o_1, está também associado a uma OrderID o_2 retornada por q_5. Posto isto, os resultados obtidos através de q_3 projetados no valor de EmployeeID e_1 conterão naturalmente valores de UnitPrice associados a o_1 e o_2.

Neste caso, é possível filtrar os valores (UnitPrice $>=$ 45). No entanto, é necessário recorrer à especificação do contexto corrente fornecida pelos resultados obtidos por q_1 e q_3. Esta especificação é dada pelos valores de todos atributos exceto os dos que estão a ser já procurados. Neste caso, o contexto corrente seria formado pelos valores dos atributos { CustomerID, CompanyName, ContactName, OrderID, EmployeeID, OrderDate}.

Agregação de múltiplas cláusulas where

No âmbito da utilização de cláusulas where na avaliação direta da política de controlo de acesso, pode-se dar o caso de se conseguir associar múltiplas cláusulas where ao contexto de um dado tuplo sensível. Perante este cenário é necessário tentar agregá-las.

Este processo de agregação consiste em tentar obter a interseção dos contextos fornecidos pelas queries. Considere-se o estado da tabela 3.4 e o conjunto de queries apresentados na listagem 4.11.

```
QILL: select EmployeeID, FirstName, LastName from Employees where Salary >
    2000;
Q1: select EmployeeID, LastName, FirstName from Employees;
Q2: select EmployeeID from Employees where Salary > 1500;
Q3: select EmployeeID from Employees where Salary < 2500;
```

Listagem 4.11: Definição de caso de uso no âmbito da agregação de cláusulas where (1)

Nesta situação pode-se perceber que através da interseção dos contextos fornecidos pelas queries q_2 e q_3 consegue-se aferir, para employees, se possuem ou não (Salary > 2000), violando assim a política de controlo de acesso estabelecida:

- como (EmployeeID = 1) e (EmployeeID = 4) são retornados por q_2 e não são retornados por q_3 pode-se concluir que possuem ((Salary > 1500) ∩ (Salary < 2500)^C) = ((Salary > 1500) ∩ (Salary >= 2500)) = (Salary >= 2500), ou seja, que os respetivos employees correspondem a tuplos sensíveis protegidos;
- como (EmployeeID = 9) é retornado por q_3 e não é retornado por q_2, consegue-se aferir que orespetivo employee possui (Salary <= 1500), ou seja, corresponde a um tuplo sensível não protegido;
- para os restantes, pode-se concluir que os respetivos Salary estão contidos no intervalo ((Salary > 1500) ∩ (Salary < 2500)).

Relativamente à implementação deste processamento, recorreu-se ao método `getAdminCount`: para o respetivo tuplo sensível, se o número de tuplos válidos em ambos os contextos for superior a zero, conclui-se que a interseção dos contextos é válida. Caso contrário sabe-se que se tratam de contextos distintos.

A aplicação desta abordagem ao exemplo apresentado no âmbito da listagem 4.11 é simples de perceber. Para o caso do empregado identificado por (EmployeeID = 1), tendo em conta a informação que deve ser fornecida ao método `getAdminCount`, atribuiu-se:

- {(EmployeeID = 1), (LastName = 'Davolio'), (FirstName = 'Nancy')} a (1);
- {(Salary > 1500), (Salary >= 2500)} a (2);
- {EmployeeID, FirstName, LastName, Salary} a (3);

Perante a informação fornecida, obtém-se o número de tuplos válidos na tabela Employees que repeitem (1) e (2). A contagem obtida corresponde a 1 > 0, ou seja, a interseção destes dois contextos é armazenada.

Esta abordagem, tal como nas restantes em que se recorre à malha inicial, possibilita que não seja necessário que efetuar um processamento adicional; neste caso, não se tem em conta se o utilizador consegue associar os respetivos contextos.

A título de exemplo, para o esquema da base de dados apresentado na figura A.1 considere-se o conjunto de queries apresentado na listagem 4.12.

```
1 Q1: select CustomerID from Orders where OrderID < 50;
2 Q2: select CustomerID from Orders where OrderID > 40;
```

Listagem 4.12: Definição de caso de uso no âmbito da agregação de cláusulas where (2)

Neste caso, pode ocorrer o caso de um utilizador estar associado a uma Order com (40 < OrderID < 50). Perante a informação obtida através de q_1 e q_2 o utilizador não consegue detetar esta situação. No entanto recorrendo à solução proposta, consegue-se aferir se este cenário se verifica ou não.

A forma como esta informação é armazenada é similar à forma como se armazenam os tuplos retornados por cada query.

Para cada contexto fornecido pelo processo de agregação das cláusulas where, é criada uma linha na coleção que armazena os resultados obtidos através dos caminhos percorridos e ao nível da mesma, armazenado o respetivo contexto, sob a forma da estrutura `Restriction` (apresentada na seção `Agregação de Expressões Simples` do Anexo `C`).

```
1 QILL: select CustomerID, CompanyName, ContactName from Customers join Orders on
      Customers.CustomerID = Orders.CustomerID join OrderDetails on Orders.
    OrderID = OrderDetails.OrderID and UnitPrice > 50.00;
2 Q1: select CustomerID, CompanyName, ContactName from Customers;
3 Q2: select CustomerID from Orders join OrderDetails on Orders.OrderID =
    OrderDetails.OrderID where UnitPrice > 40.00;
4 Q3: select CustomerID from Orders join OrderDetails on Orders.OrderID =
    OrderDetails.OrderID where UnitPrice < 30.00;
```

Listagem 4.13: Definição de caso de uso no âmbito da agregação de cláusulas where (3)

Caso a associação entre o contexto corrente e os tuplos retornados por uma query ser inválida, a respetiva informação é armazenada numa coleção que contém informação inválida. Através do exemplo apresentado no âmbito da listagem 4.13 pode-se perceber a necessidade de se utilizar esta estratégia: apesar de um dado CustomerID (cid_1) ser retornado por q_3 não se pode afirmar que o customer identificado por cid_1 não esteja associado a um (UnitPrice > 50.00), visto que pode estar associado a múltiplos UnitPrices. No entanto, se um CustomerID (cid_2) não for retornado por q_2 sabe-se que o mesmo não está associado a um (UnitPrice > 50.00). No segundo caso pode-se concluir que cid_2 não está associado a (UnitPrice > 50.00) uma vez que a associação entre o contexto dado pelos valores de { CustomerID, CompanyName, ContactName}, obtidos em q_1, e o tuplo retornado por q_2 ser inválida.

Esta abordagem permite que o processo de avaliação seja efetuado de igual forma para informação inferida através de cláusulas where e através dos tuplos devolvidos pelas queries, uma vez que a estrutura `Restriction` permite operações de validação de `SingleExpressions` e de outras `Restrictions`.

4.7.5 Agregação dos conjuntos de resultados obtidos

Tal como referido na subseção `Agregação dos Resultados Obtidos` 3.3.1, é necessário processar os resultados obtidos através de múltiplos caminhos.

A agregação dos resultados obtidos através dos múltiplos caminhos consiste na realização das operações de interseção, união e diferença dos conjuntos de resultados obtidos a partir dos múltiplos caminhos atravessados, tal como e poderá perceber ao longo desta subseção.

Considere-se um exemplo assente no caso de uso apresentado na listagem 4.14.

```
QILL: select CustomerID, CompanyName, ContactName from Customers join Orders on
    Customers.CustomerID = Orders.CustomerID join OrderDetails on Orders.
    OrderID = OrderDetails.OrderID and UnitPrice > 50.00;

Q1: select CustomerID, CompanyName, ContactName from Customers;
Q2: select OrderID, EmployeeID, CustomerID from Orders;
Q3: select OrderDate, UnitPrice from OrderDetails join Orders on OrderDetails.`
    OrderID` = `Orders`.`OrderID`;
Q4: select RequiredDate, ProductID, UnitPrice from Orders join OrderDetails on
    Orders.OrderID= OrderDetails.OrderID where EmployeeID <= 4;
Q5: select OrderID, OrderDate, RequiredDate, Employeeid from Orders;
Q6: select RequiredDate, ProductID, UnitPrice from `orders` join OrderDetails
    on orders.`OrderID`=OrderDetails.`OrderID` where EmployeeID > 5;
```

Listagem 4.14: Definição de caso de uso no âmbito da agregação de resultados obtidos

Perante este caso de uso, é possível obter possíveis valores do atributo UnitPrice associados a um Customer através dos caminhos $C_1 = \{ q_1, q_2, q_5, q_3\}$, $C_2 = \{ q_1, q_2, q_5, q_4\}$ e $C_3 = \{ q_1, q_2, q_5, q_6\}$. Considere-se que R_{C1}, R_{C2} e R_{C3} correspondem aos resultados obtidos através de C_1, C_2 e C_3, respetivamente.

Tal como é apresentado no algoritmo 5, pode-se perceber que os resultados obtidos a partir de cada caminho percorrido, vão sendo adicionados a uma coleção que armazena a agregação dos mesmos. Posto isto, considere-se R_{aggr} o resultado da agregação da informação obtida através dos múltiplos caminhos.

No âmbito deste exemplo, esta agregação é realizada em três iterações.

1. $R_{aggr1} = \{ R_{C1}\}$
2. $R_{aggr2} = \{ (R_{aggr1} - (R_{aggr1} \cap R_{C2})) \cup (R_{C2} - (R_{aggr1} \cap R_{C2})) \cup (R_{aggr1} \cap R_{C2})\} = \{ (R_{C1} - (R_{C1} \cap R_{C2})) \cup (R_{C2} - (R_{C1} \cap R_{C2})) \cup (R_{C1} \cap R_{C2})\}$
3. $R_{aggr3} = \{ (R_{aggr2} - (R_{aggr2} \cap R_{C3})) \cup (R_{C3} - (R_{aggr2} \cap R_{C3})) \cup (R_{aggr2} \cap R_{C3})\}$

Adicionalmente, no âmbito da subeção `Agregação de resultados obtidos` 3.3.1, pôde-se também perceber que a interseção não é sempre realizada:

- os contextos em que os valores dos atributos sensíveis de predicado são obtidos podem não se intersetar, como é apresentado no exemplo formulado no âmbito da listagem 3.9;
- apesar dos contextos em que os valores dos atributos sensíveis de predicado são obtidos se intersetarem, o utilizador pode não ter tido acesso a informação suficiente para efetuar essa associação, tal como ficou patente no exemplo apresentado no âmbito da listagem 3.10[5].

No âmbito deste exemplo, através da análise das cláusulas where de q_4 e q_6 pode-se perceber que $(R_{C2} \cap R_{C3}) = \varnothing$.

Tendo em conta as cláusulas where das queries q_3, q_4 e q_6, a figura 4.8 pode corresponder a uma representação dos resultados obtidos (que são efetivamente associados a um dado Customer) no âmbito dos três caminhos supramencionados.

[5]De fato, para este exemplo, tendo em conta a solução apresentada na subseção 4.7.4, pode-se perceber que a mesma consegue efetuar algumas associações mesmo quando o utilizador não o consegue efetuar.

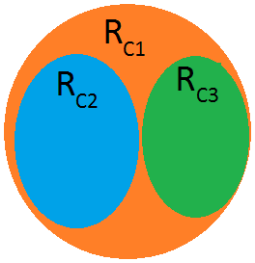

Figura 4.8: Representação de R_{C1}, R_{C2} e R_{C3}

Posto isto, a forma como se obtém a interseção entre dois caminhos distintos passa por formar um caminho C_{aggr} que contenha as queries de ambos. Posteriormente, o processo que simboliza o fato de um utilizador conseguir efetuar a associação entre os contextos dos resultados obtidos no âmbito dos dois caminhos corresponde simplesmente a atravessar C_{aggr}. É fácil de perceber que, apesar de esta abordagem ser fácil de perceber concetualmente, exige um processamento adicional significativo.

A título de exemplo, considere-se que Q_i, Q_{ii} e Q_{iii} correspondem ao conjunto de queries que compõem os caminhos C_i, C_{ii} e C_{iii}, respetivamente. No limite, na terceira iteração do ciclo `foreach` do algoritmo 5, pode-se perceber que são atravessados três caminhos. $C_{aggr3.1}$ formado por Q_i e Q_{iii}; $C_{aggr3.2}$ formado por Q_{ii} e Q_{iii}; $C_{aggr3.3}$ formado por Q_i, Q_{ii} e Q_{iii}.

Voltando novamente ao exemplo apresentado no âmbito da listagem 4.14, os possíveis caminhos atravessados no âmbito da terceira iteração correspondem a:

- $C_{aggr3.1} = (Q_{C1} \cup Q_{C3}) = \{ q_1, q_2, q_5, q_3, q_6\}$;
- $C_{aggr3.2} = (Q_{C2} \cup Q_{C3}) = \{ q_1, q_2, q_5, q_4, q_6\}$;
- $C_{aggr3.3} = (Q_{C1} \cup Q_{C2} \cup Q_{C3}) = \{ q_1, q_2, q_5, q_3, q_4, q_6\}$.

Como referido, o processamento descrito no algoritmo 5 é efetuado para obter valores de atributos associados ao contexto corrente para (1) posterior avaliação e (2) para filtragem dos tuplos obtidos numa query no âmbito de um caminho. Assim, pode-se perceber que em certos cenários não é necessário atravessar todos os possíveis caminhos para obter os resultados necessários à execução de (1) e (2).

Consequentemente, para o caso de (1) considera-se que não é necessário atravessar mais caminhos se

- os resultados obtidos no processo de agregação dos caminhos já atravessados permitam concluir que o contexto corrente é válido no âmbito da cláusula where que se pretende avaliar;
- os resultados obtidos no processo de agregação dos caminhos já atravessados permitam concluir que o contexto corrente é inválido no âmbito da cláusula where que se pretende avaliar e que foram obtidos todos os valores no âmbito do respetivo contexto.

Por outro lado, se se pretende efetuar (2), a solução desenvolvida considera que não é necessário atravessar mais caminhos caso se tenham obtido todos os valores dos atributos no

âmbito do contexto corrente e seja possível perceber que eles estão efetivamente associados a respetivo contexto; ou seja, através de resultados do tipo REL.

Adicionalmente, no âmbito do atravessamento de C_{aggr}, enquanto um mesmo caminho (C_1) contiver as queries de C_{aggr}, não é necessário atravessá-las da forma *tradicional* e, consequentemente, de utilizar recursividade quer para avaliar as cláusulas where das queries quer para filtrar possíveis tuplos inválidos devolvidos pelas mesmas. Neste caso, como resultado do atravessamento de C_{aggr} corresponderão a ($R_{C1} \cap R_{C2}$), sabe-se que os resultados presentes no sub-caminho de C_1 contêm todos os resultados de R_{aggr}.

Esta abordagem não pode ser realizada no contexto do atravessamento de cada caminho individualmente, visto que os contextos dos mesmos tipicamente não são os mesmos. Os caminhos $C_2 = \{ q_1, q_2, q_5, q_4\}$ e $C_3 = \{ q_1, q_2, q_5, q_6\}$ apresentados no âmbito da listagem 4.14, demonstram esta justificação. Apesar de as primeiras três queries de C_2 e C_3 serem iguais, como os contextos de q_4 e q_6 nem se intersetam, as Orders armazenadas nas coleções que contêm os resultados destes dois caminhos são distintas.

Ordenação de queries na formação de C_{aggr}

Ao nível da formação destes caminhos, foi necessário ordenar as queries que os formam recorrendo a um processo iterativo. Neste processo, a cada iteração é escolhida a próxima query a adicionar a C_{aggr} de entre a próxima query que forma cada um dos respetivos caminhos. Este processo de decisão tenta minimizar, a cada passo a informação necessária de processar. Para tal utiliza vários critérios para tentar decidir; optando, em última instância, aleatoriamente por uma das queries possíveis.

O primeiro critério utilizado passa por minimizar as tabelas atravessadas até ao próximo passo do caminho. Considere-se os dois caminhos $C_1 = \{ q_1, q_2, q_3\}$ e $C_2 = \{ q_1, q_2, q_4, q_5\}$ formados pelas queries apresentadas na listagem 4.15.

```
1 Q1: select CustomerID, CompanyName, ContactName from Customers;
2 Q2: select OrderID, EmployeeID, CustomerID from Orders;
3 Q3: select EmployeeID, OrderDate, UnitPrice from Orders join OrderDetails on
    Orders.OrderID = OrderDetails.OrderID;
4 Q4: select ShippedDate, EmployeeID from Orders where OrderDate > '1998-01-05';
5 Q5: select ShippedDate, UnitPrice from Orders join OrderDetails on OrderDetails
    .`OrderID` = `Orders`.`OrderID`;
```

Listagem 4.15: Definição de caso de uso no âmbito da ordenação de queries de C_{aggr} (1)

Nas primeiras iterações escolhem-se as queries q_1 e q_2. Posteriormente, a escolha recairá sobre q_3 ou q_4. De acordo com os atributos obtidos até este momento, foram inquiridos atributos pertencentes às tabelas Customers e Orders. Adicionalmente, pode-se constatar que q_4 inquire apenas atributos da tabela Orders, enquanto que q_3 inquire um atributo da tabela OrderDetails. Posto isto, de forma a que não seja necessário atravessar atributos da tabela OrderDetails no passo seguinte, opta-se por q_4.

O segundo critério baseia-se nas restrições da cláusula where. No âmbito deste critério tenta-se que a escolha recaia sobre a query cujo contexto da cláusula where seja englobado pelo contexto da cláusula where da outra query. Ou seja, para duas queries que inquiram os mesmos

atributos na cláusula select, cujas cláusulas where são definidas por R_1 = (EmployeeID > 5) e R_2 = (EmployeeID > 10), opta-se pela query com a cláusula where R_2, visto que $R_2 \subset R_1$.

No limite, o contexto de uma cláusula where nula engloba o contexto de todos os outros contextos.

O terceiro e último critério baseia-se nos atributos inquiridos na cláusula select das queries. Neste contexto opta-se pela query cujo número de atributos inquiridos que pertencem ao conjunto $C_{not\_all\_rel}$ (tal como foi apresentado no âmbito da subseção `Atravessar um caminho` 4.7.3), e consequentemente têm que ser obtidos recursivamente, for menor.

Considerem-se os caminhos $C_1 = \{ q_1, q_2, q_3\}$ e $C_2 = \{ q_1, q_2, q_4\}$ formados pelas queries apresentadas na listagem 4.16. Neste cenário, após obter as queries q_1 e q_2 nas duas primeiras iterações, a escolha recairá sobre uma das queries q_3 e q_4. Perante este critério, opta-se por q_4, visto que $C_{not\_all\_rel}(q_3) = \{$ OrderDetails.UnitPrice$\}$ enquanto que $C_{not\_all\_rel}(q_4) = \varnothing$.

```
1 Q1: select CustomerID, CompanyName, ContactName from Customers;
2 Q2: select OrderID, EmployeeID, CustomerID from Orders;
3 Q3: select EmployeeID, UnitPrice from Orders join OrderDetails on Oders.OrderID
      = OrderDetails.OrderID;
4 Q4: select OrderID, UnitPrice from Orders join OrderDetails on Oders.OrderID =
      OrderDetails.OrderID;
```

Listagem 4.16: Definição de caso de uso no âmbito da ordenação de queries de C_{aggr} (2)

Agregação de múltiplos caminhos recorrendo a resultados inválidos

No âmbito de processamento dos resultados obtidos através de múltiplos caminhos, surge o conceito de resultados inválidos.

Tal como foi descrito no âmbito desta solução, o objetivo passa por obter conjuntos de possíveis valores associados a um dado tuplo sensível para, posteriormente, proceder à sua avaliação. Neste contexto, é intuitivo pensar que apenas interessam os resultados obtidos no âmbito de caminhos válidos. No entanto, em certas situações, é possível filtrar resultados inválidos de conjuntos de resultados NAR recorrendo precisamente a conjuntos de resultados inválidos.

Considerem-se os caminhos $C_1 = \{ q_1, q_2, q_3\}$ e $C_2 = \{ q_1, q_2, q_4\}$ formados pelas queries apresentadas na listagem 4.17.

```
1 Q1: select CustomerID, CompanyName, ContactName from Customers;
2 Q2: select OrderID, EmployeeID, CustomerID from Orders;
3 Q3: select EmployeeID, UnitPrice from Orders join OrderDetails on Orders.
      OrderID = OrderDetails.OrderID;
4 Q4: select EmployeeID, OrderDate, UnitPrice from Orders join OrderDetails on
      Orders.OrderID = OrderDetails.OrderID where Orders.OrderID < 100;
```

Listagem 4.17: Definição de caso de uso no âmbito de agregação de resultados válidos com inválidos

No âmbito do caminho C_1 pode-se perceber que o último passo é do tipo NAR. Para uma Order com (Orders.OrderID < 100), através de C_2, podem-se filtrar valores inválidos obtidos no âmbito de C_1, ao realizar a interseção destes dois caminhos, tal como descrito nesta subseção. No entanto, para uma Order com (Orders.OrderID >= 100), de acordo com

o que foi descrito até este momento, esta solução não permite filtrar tuplos inválidos obtidos no âmbito de C_1, que um utilizador pode filtrar recorrendo aos resultados obtidos através de C_2.

Posto isto, pode-se perceber que além de percorrer os caminhos em que obtém tipos de resultados validos (REL e NAR), é necessário também percorrer caminhos em que o tipo de resultados final é inválido (INV). De fato, pode-se perceber que no âmbito de um caminho, um passo em específico pode conter tipos de resultados REL, NAR e INV.

Um exemplo deste caso corresponde ao que acontece em C_2. Dependendo do estado da base de dados, pode ocorrer o caso de um CustomerID:

- estar associado a uma OrderID ($o_{id1} < 100$) cujo EmployeeID e_{id1} apenas está associado a o_{id1}; fazendo com que os resultados obtidos no âmbito de q_4 sejam do tipo REL;
- estar associado a uma OrderID ($o_{id2} < 100$) cujo EmployeeID e_{id2} está associado a múltiplas Orders; fazendo com que os resultados obtidos no âmbito de q_4 sejam do tipo NAR;
- estar associado a uma OrderID ($o_{id3} >= 100$) cujo EmployeeID e_{id3} está associado a múltiplas Orders, sendo que pelo menos existe uma ($o_{idN} < 100$); fazendo com que os resultados obtidos no âmbito de q_4 sejam do tipo INV.

De fato, quando se percorre um caminho, apenas se armazena um resultado INV caso a query que o origina seja a última do caminho; caso contrário os valores obtidos são descartados.

Exemplificando, para o exemplo apresentado no âmbito da listagem 4.17, C_{aggr} = { q_1, q_2, q_3, q_4}. Neste caso, a figura 4.9 representa o resultado de C_{aggr}.

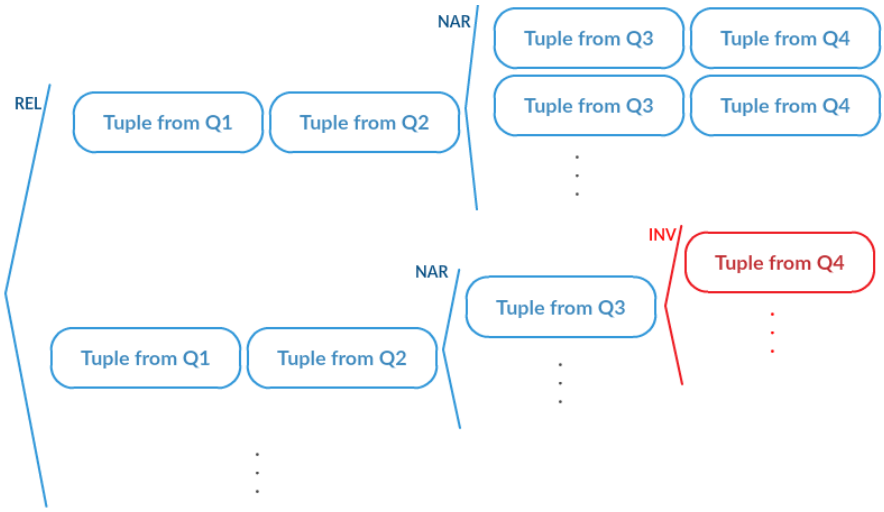

Figura 4.9: Resultado do processamento de C_1 e C_2

Através da análise desta figura pode-se perceber que a primeira linha da coleção REL contém uma (OrderID < 100), visto que os tuplos de q_3 são depois associados aos de q_4. Por outro lado, a segunda linha está associada a uma (OrderID >= 100). Neste caso, recorrendo ao método `getAdminCount`, descrito inicialmente na subseção 4.7.1, verifica-se se se pode remover os tuplos de q_3 associados à segunda linha da coleção REL. Esta análise consiste

simplesmente em comparar o número de ocorrências dos valores dos atributos retornados por q_3 (query que retorna o tipo de resultados válido) no contexto da cláusula where de q_3, $count_3$, com o número de ocorrências dos valores dos atributos retornados por q_4 (query que retorna o tipo de resultados inválido), também no contexto da cláusula where de q_4, $count_4$. Apenas se ($count_4 = count_3$) se pode considerar o respetivo tuplo de q_3 inválido.

Registo da utilização de tuplos de uma coleção

Tal como explicado nesta subseção, é necessário manter, aquando do processamento de múltiplos caminhos, os resultados obtidos no âmbito das iterações anteriores.

No entanto, após esta fase não é desejável ter acesso à mesma informação múltiplas vezes:

1. tal como mencionado no início desta subseção 4.7.5, na obtenção dos valores de mais do que um atributo, os valores dos seguintes são obtidos recursivamente para cada especificação do contexto corrente. Quantas mais especificações houverem neste contexto, maior o número de chamadas recursivas necessárias para obter os valores associados ao contexto corrente;

2. através do processamento descrito ao longo da subseção 4.7.5 é possível excluir valores inválidos de atributos em coleções do tipo NAR, o que na maioria dos casos faz a diferença no processo de avaliação;

3. no âmbito do atravessamento de um caminho, na filtragem dos valores dos atributos que compõem ao conjunto $C_{not\_all\_rel}$, a presença dos valores dos respetivos atributos na coleção onde são armazenados os resultados obtidos indica que o utilizador, através das queries que executou, os percebe como possíveis valores válidos. Logo, tuplos que sejam compostos pelos mesmos são adicionados incorretamente aos resultados obtidos no âmbito desse passo.

Perante este cenário, torna-se necessária uma solução que permita saber quando uma linha de uma coleção está presente no resultado da interseção/ diferença com outro caminho. Posteriormente, recorrendo a esta sinalização, podem-se descartar linhas utilizadas múltiplas vezes (tal como é descrito em `Remoção de resultados sobrepostos` 4.7.6).

Posto isto, numa primeira abordagem poder-se-ia ponderar registar esta reutilização através de uma *flag* que indicasse que a respetiva linha tinha sido reutilizada. No entanto, tendo em conta a solução para remover linhas reutilizadas no âmbito do processo de agregação com outros caminhos (que será descrita em 4.7.6), é fácil perceber que esta abordagem seria errada.

Recorrendo ao exemplo baseado nas queries apresentadas na listagem 4.18, pode-se perceber que é possível obter os valores do atributo `UnitPrice` através de dois caminhos:

- $C_1 = \{q_1, q_2, q_3\}$;
- $C_2 = \{q_1, q_4, q_5\}$;

```
1 QILL: select CustomerID, CompanyName, ContactName from Customers join Orders on
      Customers.CustomerID = Orders.CustomerID join OrderDetails on Orders.
    OrderID = OrderDetails.OrderID and UnitPrice > 50.00;
2
```

```
Q1: select CustomerID, CompanyName, ContactName from Customers;
Q2: select CustomerID, RequiredDate from Orders where EmployeeID < 4;
Q3: select RequiredDate, UnitPrice from Orders;
Q4: select CustomerID, OrderDate from Orders;
Q5: select OrderDate, UnitPrice from Orders;
```

Listagem 4.18: Definição de caso de uso no âmbito do registo de (re)utilização de linhas de uma coleção

Tendo em conta os atributos inquiridos nas queries que compõem C_1, tipicamente a topologia dos resultados obtidos no âmbito do respetivo caminho corresponde à apresentada na figura 4.10; sendo a topologia dos resultados obtidos através de C_2 similar.

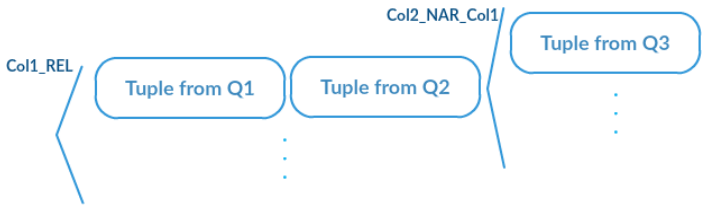

Figura 4.10: Representação da topologia dos resultados obtidos através de C_1

OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate
1	cid1	3	date1	date2
2	cid1	8	date1	date2
...	...	...	...	...

Tabela 4.1: Exemplo do contexto da tabela Orders (1)

No contexto apresentado na tabela 4.1, suponha-se que as Orders identificadas por (OrderID = 1) e (OrderID = 2) estão associadas a um mesmo valor de (UnitPrice = P). Para (CustomerID = 'cid1'), quando a query q_4 é executada aquando do atravessamento de C_2, um dos tuplos obtidos é dado por t_1 = {CustomerID = 'cid1', OrderDate = 'date1'}. Neste caso, t_1 está associado às Orders identificadas por (OrderID = 1) e (OrderID = 2).

No âmbito do processo de agregação dos resultados obtidos através de C_1 e C_2, é fácil perceber que através de C_{aggr} obter-se-á também o tuplo t_1. No entanto, este mesmo tuplo está apenas associado a (OrderID = 1). Perante este caso, pode-se perceber que apesar de t_1 ser obtido no âmbito de C_{aggr}, aquando da remoção dos tuplos reutilizados, não se pode remover a respetiva linha da coleção obtida no âmbito de C_2. Se fosse utilizada apenas uma *flag* para indicar se a linha tinha ou não sendo reutilizada, não seria possível detetar este caso.

Posto isto, para registar a reutilização de uma linha de uma coleção obtida através do atravessamento de um caminho C_i, passou por registar, no âmbito da respetiva linha, o contexto em que os tuplos que a compõem são reutilizados no processo de agregação com um novo caminho. Este contexto é dado pelas cláusulas where das queries que compõem cada linha das coleções obtidas no âmbito de C_{aggr}.

Para cada linha L_i de uma coleção obtida através de um caminho (composta pelo conjunto de tuplos T_i das respetivas queries), o contexto em que a mesma é reutilizada é dado pelas

cláusulas where das queries que compõem as linhas das coleções atravessadas, no âmbito de C_{aggr}, até ao último tuplo pertencente a T_i.

A justificação para se efetuar o registo da reutilização desta forma prende-se com o fato de se saber que a informação armazenada no âmbito de uma linha da coleção (os valores dos atributos e o contexto em que eles foram obtidos) está associada entre si. Posto isto, sabe-se que o contexto (Ctx_{aggr}) de cada linha pertencente a uma coleção resultante da agregação de dois caminhos onde aparece pelo menos um tuplo pertencente à linha da coleção *original*, está incluído no contexto (Ctx_{orig}) desse mesmo tuplo: $Ctx_{aggr} \subset Ctx_{orig}$.

Este registo é efetuado recorrendo à estrutura `SingleExpression` armazenado em:

- <attribute_name> a macro `OVERLAPPED`;
- <relation> `EQUALS`;
- <attribute_value> os identificadores de cada query que compõe Ctx_{aggr}.

Recorrendo ao exemplo utilizado anteriormente, para (CustomerID = 'cid1'), no âmbito da especificação supramencionada obtida através de C_1, os respetivos registos da reutilização das linhas das coleções obtidas são apresentados na figura 4.11.

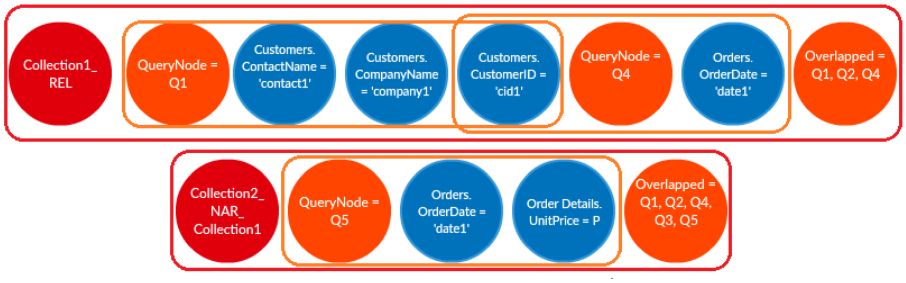

Figura 4.11: Representação dos registos de reutilização de linhas das coleções obtidas através de C_1

Neste contexto, convém mencionar que a representação não está efetivamente correta na medida em que o nó de coleção identificado por `Collection2_NAR_Collection1` deveria ser abrangido pela pela linha de cima e o nó `Orders.OrderDate = 'date1'` deveria ser reutilizado.

4.7.6 Remoção de resultados sobrepostos

Como se pôde perceber ao longo da subseção anterior, no âmbito da reutilização de uma linha de uma coleção, resultante do atravessamento de um caminho, é necessário registar o contexto em que esta situação ocorre. Este registo é utilizado posteriormente com o intuito de ao atravessar as coleções que contêm os resultados obtidos, remover os conjuntos de resultados repetidos.

Posto isto, utilizando uma aproximação *bottom-up*, começar-se-á por descrever a forma como se determina se se pode ou não remover uma linha de uma coleção. Posteriormente, dependendo das linhas efetivamente removidas de uma coleção, verifica-se se se pode ou não

eliminar a própria coleção.

No âmbito do exemplo apresentado na subseção anterior, foi possível perceber que para (CustomerID = 'cid1'), apesar de alguns dos tuplos obtidos através de C_2 serem reutilizados no âmbito C_{aggr}, os mesmos não podiam ser removidos. Isto devia-se ao fato de o contexto em que os respetivos tuplos eram reutilizados no âmbito de C_{aggr} não abranger o contexto em que e eram obtidos em C_2.

Neste caso, para que se pudesse remover os respetivos tuplos, seria necessário que adicionalmente fossem executadas queries que garantissem que todo o contexto tinha sido abrangido. Posto isto, considere-se uma extensão do exemplo supramencionado baseada na listagem 4.19.

```
1 QILL: select CustomerID, CompanyName, ContactName from Customers join Orders on
      Customers.CustomerID = Orders.CustomerID join OrderDetails on Orders.
     OrderID = OrderDetails.OrderID and UnitPrice > 50.00;
2
3 Q1: select CustomerID, CompanyName, ContactName from Customers;
4 Q2: select CustomerID, RequiredDate from Orders where EmployeeID < 4;
5 Q3: select RequiredDate, UnitPrice from Orders;
6 Q4: select CustomerID, OrderDate from Orders;
7 Q5: select OrderDate, UnitPrice from Orders;
8 Q6: select select CustomerID, RequiredDate from Orders where EmployeeID between
      3 and 6;
9 Q7: select CustomerID, RequiredDate from Orders where EmployeeID >= 5;
```

Listagem 4.19: Definição de caso de uso no âmbito da remoção de linhas de coleção (re)utilizadas em múltiplas coleções

Perante este caso, pode-se perceber que adicionalmente seria possível obter valores de UnitPrice através dos caminhos:

- $C_3 = \{q_1, q_6, q_3\}$;
- $C_4 = \{q_1, q_7, q_3\}$;

Posto isto, é fácil depreender que, para o caso de (CustomerID = 'cid1'), os respetivos resultados obtidos através de C_2, seriam reutilizados no âmbito do processo de agregação com os caminhos C_3 e C_4. Ou seja, no âmbito da linha da coleção `Collection1_REL` apresentada na figura 4.11, adicionalmente estariam dois nós definidos por:

- Overlapped = $\{q_1, q_6, q_4\}$
- Overlapped = $\{q_1, q_7, q_4\}$

Neste cenário, pode-se perceber que os contextos nos quais os respetivos tuplos tinham sido reutilizados eram dados por Ctx_{aggr1} = { (EmployeeID < 4)}, Ctx_{aggr2} = { (EmployeeID between 3 and 6)} e Ctx_{aggr3} = { (EmployeeID >= 5)}. Deste modo, poder-se-ia então remover as respetivas linhas da coleção resultante do atravessamento do caminho C_2, uma vez que a reunião de Ctx_{aggr1}, Ctx_{aggr2} e Ctx_{aggr3} abrangeria o contexto de C_2.

Neste exemplo, foi possível perceber que a união dos contextos (Ctx_{un}), definidos pelas cláusulas where das queries q_2, q_6 e q_7, abrangia o contexto de C_2 (Ctx_2) apenas através

da análise das respetivas cláusulas. No entanto, podem ser realizadas queries cujas cláusulas where não incidam sempre sobre os mesmos atributos. Considere-se que o estado de uma base de dados corresponde ao apresentado na tabela 4.1 e que são executadas as queries apresentadas na listagem 4.20.

```
1 Q1: select * from Orders where EmployeeID < 4;
2 Q2: select * from Orders where OrderID >= 2;
```

Listagem 4.20: Definição de queries no âmbito da agregação de contextos

Perante este exemplo pode-se perceber que q_1 e q_2 inquirem a totalidade da informação armazenada na respetiva tabela. No entanto, esta perceção não é obtida através da exclusiva análise das cláusulas where. Para tal, é necessário também de analisar a informação armazenada na base de dados. Esta necessidade implicaria *descobrir* se o utilizador tinha tido acesso a informação suficiente para inferir que toda a informação contida na tabela 4.1 era obtida através das queries q_1 e q_2.

Posto isto, para avaliar se Ctx_{un} abrange Ctx_i, utiliza-se novamente o paradigma *soft computing*. De fato, a forma como se efetua este processamento não recorre à análise das cláusulas where mas sim aos resultados efetivamente obtidos nos respetivos contextos. Nesta abordagem, não se tem em conta a informação a que o utilizador teve acesso, ou seja, sempre que se verificar que Ctx_{un} abrange Ctx_i, independentemente do utilizador ter essa perceção, remove-se a respetiva linha da coleção. A forma como se verifica se Ctx_{un} abrange Ctx_i é apresentada na seção `Algoritmo de verificação da necessidade de remoção de linhas sobrepostas de coleções` do Anexo `E`. Nela é também apresentada a forma como, através do algoritmo descrito, se consegue detetar que para o exemplo estendido revelado na listagem 4.19, a linha da coleção `Collection1_REL`, apresentada na figura 4.11, necessita de ser removida.

Por fim, tal como mencionado no início desta subseção, após verificar se se pode remover cada linha, é necessário também verificar se se pode remover a coleção em si. A forma como se efetuar este processamento é descrita na seção `Algoritmo de remoção de resultados sobrepostos` do Anexo `E`.

4.7.7 Remoção adicional de tuplos inválidos

Como foi possível perceber na subseção `Processamento dos resultados obtidos` 4.7.5 é possível percecionar, no âmbito das operações de interseção e diferença entre conjuntos de resultados obtidos, tuplos inválidos no contexto de um dado tuplo sensível, podendo posteriormente descartá-los. Esta perceção efetua-se recorrendo a associações (projeções) entre valores de atributos que são válidos, ou que, pelo menos, são *percebidos* como possivelmente válidos (este corresponde ao princípio de inferência base da solução desenvolvida).

No entanto, é também possível um utilizador perceber que determinada informação é inválida num dado contexto se a conseguir associar a valores de atributos inválidos no respetivo contexto, ou seja, se a projeção for efetuada sobre valores de atributos inválidos.

Considere-se os estados das tabelas Orders e Customers apresentados nas tabelas 4.2 e 4.3, respetivamente. Adicionalmente, considere-se um caso de uso composto pelas queries presentes na listagem 4.21.

OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate
1	cid1	3	date1	date2
2	cid2	8	date1	date2
3	cid1	9	date4	date3
...	...	...	...	...

Tabela 4.2: Exemplo do contexto da tabela Orders (2)

CustomerID	CompanyName	ContactName	Country	
cid1	company1	contact1	USA	
cid2	company2	contact2	USA	
...	...	...	...	...

Tabela 4.3: Exemplo do contexto da tabela Customers (1)

```
QILL: select CustomerID, CompanyName, ContactName from Customers join Orders on
    Customers.CustomerID = Orders.CustomerID join OrderDetails on Orders.
    OrderID = OrderDetails.OrderID and UnitPrice > 50.00;

Q1: select CustomerID, CompanyName, ContactName, Country from Customers;
Q2: select Country, OrderID from Orders join Customers on Orders.CustomerID =
    Customers.CustomerID;
Q3: select CustomerID, OrderID from Orders where OrderID <= 2;
Q4: select OrderID, UnitPrice from OrderDetails;
```

Listagem 4.21: Definição de caso de uso no âmbito da remoção adicional de tuplos inválidos

Neste contexto, pode-se perceber que é possível obter possíveis valores válidos do atributo UnitPrice através do caminho $C_1 = \{ q_1, q_2, q_4\}$. No âmbito da execução de q_2, percebe-se que os resultados obtidos através da mesma são do tipo NAR. Perante este cenário, tal como referido na subseção `Atravessar um Caminho` 4.7.3, é necessário filtrar os tuplos obtidos através de q_2. Esta filtragem efetuar-se-á com base nos valores de OrderID percecionados como possivelmente válidos.

Para o tuplo sensível identificado por (CustomerID = 'cid1'), através de q_2 obtém-se possíveis valores válidos de OrderID através de uma projeção sobre (Country = 'USA'): {(Country = 'USA', OrderID = 1); (Country = 'USA', OrderID = 2); (Country = 'USA', OrderID = 3); ...}.

Adicionalmente, através de q_3 sabe-se que a Order identificada por (OrderID = 1) está efetivamente associada ao CustomerID supramencionado. Por outro lado, também através de q_3, pode-se perceber que a Order identificada por (OrderID = 2) está associada a (CustomerID = 'cid2'). Como esta associação é efetuada recorrendo a um valor inválido de CustomerID, o que vai contra o supramencionado princípio base de inferência da solução desenvolvida, a mesma, de acordo com o descrito até aqui, não consegue detetar esta situação.

A solução encontrada para resolver este problema passou por, numa primeira fase, para cada valor inválido de um atributo no âmbito de um dado contexto (Ctx_1), obter as queries em que é possível perceber que esse mesmo valor está associado a um contexto distinto (Ctx_2);

pode-se perceber que um atributo está associado a um contexto distinto de Ctx_1 quando o mesmo está apenas associado a valores de atributos inválidos no âmbito do respetivo contexto. Na segunda fase, se a união dos contextos (Ctx_{un}), em que se percebe que o respetivo valor é inválido, abranger Ctx_1 ($Ctx_1 \subset Ctx_{un}$), pode-se descartá-lo do conjunto de resultados obtidos.

A implementação desta solução concetual é efetuada em duas fases. A primeira corresponde a, para cada query em que o respetivo valor do atributo é retornado, perceber se o mesmo está associado apenas a valores inválidos de atributos no âmbito de Ctx_1. Se esta situação se verificar, considera-se o contexto da respetiva query na segunda fase.

No âmbito da segunda fase, onde verifica-se se a união dos contextos obtidos na primeira fase abrange Ctx_1, a solução utilizada corresponde à descrita no âmbito da subseção `Remoção de resultados sobrepostos` 4.7.6.

Como se pode perceber, este processamento pode tornar-se pesado computacionalmente na medida em que é necessário, para cada query que se pretende verificar na primeira fase, obter os valores dos atributos por ela inquiridos efetivamente associados a Ctx_1; sendo que esta obtenção dos valores dos atributos é efetuada recorrendo a recursividade.

Recorrendo ao exemplo apresentado nesta subseção, para (CustomerID = 'cid1'), quando se percorre C_1, aquando da execução de q_2, o contexto corrente é composto por Ctx_1 = { CustomerID = 'cid1', CompanyName = 'company1', ContactName = 'contact1', Country = 'USA'}. Ao verificar que o tuplo (Country = 'USA', OrderID = 2), retornado por q_2, é inválido em Ctx_1, tenta-se obter as queries em que (OrderID = 2) seja retornado e que não esteja associado a nenhum valor de atributo possivelmente válido em Ctx_1. Através da análise à informação retornada por q_3, verifica-se que (OrderID = 2) é retornado na respetiva query e que a mesma também inquire o atributo CustomerID. Como Ctx_1 já contém o valor de CustomerID, não é necessário procurar recursivamente por ele. Como não é retornado nenhum tuplo composto por { (CustomerID = 'cid1'), (OrderID = 2)}, pode-se perceber que, no contexto de q_3 é possível que um utilizador perceba que (OrderID = 2) não está associado a Ctx_1.

Por fim, afere-se que o contexto de q_3, para (OrderID = 2), abrange o contexto de q_2; uma vez que o número de tuplos válidos nos contextos de q_3 e q_2 é igual.

4.7.8 Normalização dos Resultados Obtidos

A normalização dos resultados obtidos processados corresponde ao último passo no âmbito do processo de obtenção de valores de atributos. O objetivo passa por organizar a informação obtida.

No âmbito da subseção `Processamento dos resultados obtidos` 4.7.5 a cada iteração i são criadas coleções adicionais que possibilitam o armazenamento dos resultados obtidos no âmbito de C_i e das iterações anteriores: $R_{aggr<i>} = R_{Ci} \cup (R_{Ci} \cap R_{aggr<i-1>}) \cup R_{aggr<i-1>}$. Apesar de no processo descrito em `Remoção de resultados sobrepostos` 4.7.6, se removerem os resultados repetidos e se eliminarem as coleções que após a primeira remoção apenas contêm linhas com informação inválida, é necessário efetuar um processamento adicional.

Entendendo a organização das coleções criadas como uma árvore, tal como é representado na figura 4.4, com este processamento pretende-se agregar a informação de forma a que a árvore seja a menor possível.

No âmbito do processo de agregação de resultados obtidos através de dois caminhos distintos é possível que o resultado do respetivo processamento seja composto apenas por informação associada ao contexto corrente e que esteja armazenado numa coleção do tipo NAR. Tendo em conta o estado da tabela 3.4, suponha-se que é permitido a um utilizador executar as queries apresentadas na listagem 4.22.

```
Q1: select EmployeeID, FirstName, LastName from Employees;
Q2: select LastName, FirstName, Salary from Employees;
Q3: select EmployeeID, Country from Employees;
Q4: select Salary, Country from Employees;
```

Listagem 4.22: Definição de queries no âmbito da normalização de coleções

Perante este caso de uso, pode-se perceber que é possível um utilizador associar um empregado ao seu possível salário através dos caminhos:

- $C_1 = \{q_1, q_2\}$;
- $C_2 = \{q_1, q_3, q_4\}$;

Para o empregado identificado por (EmployeeID = 9) os resultados obtidos no âmbito dos dois caminhos são do tipo NAR. No processo de agregação, de acordo com o que foi descrito na subseção `Ordenação de queries na formação de` C_{aggr} 4.7.5, $C_{aggr} = \{q_1, q_3, q_2, q_4\}$.

No âmbito da execução de q_2 aquando do atravessamento de C_{aggr}, obtém-se o conjunto de tuplos {(LastName = 'Dodsworth', FirstName = 'Nancy', Salary = 3119.15), (LastName = 'Dodsworth', FirstName = 'Nancy', Salary = 1333.33)} que é armazenado numa coleção do tipo NAR. No entanto, aquando da execução de q_4 o tuplo (LastName = 'Dodsworth', FirstName = 'Nancy', Salary = 3119.15) retornado por q_2 é descartado e a respetiva coleção NAR passa a ser composta apenas pela linha que contém os tuplos { (LastName = 'Dodsworth', FirstName = 'Nancy', Salary = 1333.33), (Coutry = 'UK', Salary = 1333.33)}.

Neste caso é fácil de perceber que o utilizador consegue perceber que (Salary = 1333.33) corresponde efetivamente ao valor associado ao empregado identificado por (EmployeeID = 9), uma vez que existe apenas um Salary associado a cada empregado. Com o processamento efetuado no âmbito desta subseção pretende-se traduzir esse conhecimento.

A forma como se implementa esta funcionalidade é fácil de perceber. Para cada coleção percorrem-se todas as linhas: se se verificar que todas são válidas no contexto em que a respetiva coleção surge, migra-se a informação nela contida para a coleção pai e apaga-se a coleção; caso contrário, mantém-se a topologia da estrutura.

Analisa-se se o conteúdo de uma linha é válido no contexto em que a coleção foi criada recorrendo ao método `getAdminCount`: se o contador de tuplos válidos para o contexto corrente e a sua especificação (dada pela respetiva linha) for maior do que zero, sabe-se que a linha é válida no âmbito do respetivo contexto. Caso contrário, a informação nela armazenada é inválida no âmbito do contexto corrente.

Este processamento começa a ser efetuado a partir das coleções *folhas*, sob a forma *post-order*[6]: antes de se verificar se todas as linhas de uma coleção Col_{raiz} são válidas no contexto em que a mesma surge, é necessário processar as coleções filhas Col_{folha}. Assim, caso uma

[6]https://en.wikipedia.org/wiki/Tree_traversal

Col_{folha} contenha apenas informação válida no contexto em que ela surge, a informação é migrada para Col_{raiz} e, aquando da validação da informação contida em Col_{raiz} é já utilizada a informação proveniente de Col_{folha}.

De fato, pode-se perceber que a forma como este processamento foi implementado, faz com que a solução desenvolvida no âmbito desta dissertação, em certos cenários, assuma incorretamente que um utilizador sabe que a informação está associada ao contexto corrente.

Para o exemplo supramencionado, para (EmployeeID = 9), as representações de como informação é armazenada antes e após o processo de normalização são apresentadas nas figuras 4.12 e 4.13, respetivamente. Pode-se constatar que após o processo de normalização a informação fica apenas armazenada numa coleção do tipo REL.

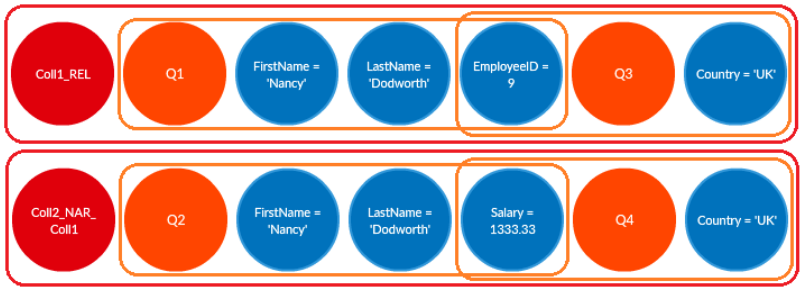

Figura 4.12: Representação do armazenamento da informação antes do processo de normalização

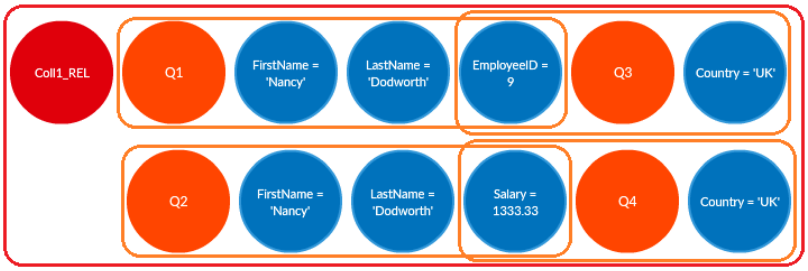

Figura 4.13: Representação do armazenamento da informação após o processo de normalização

4.7.9 Problema da recursividade

Tal como foi referido ao longo deste capítulo, esta solução utiliza recursividade em diversas situações:

- avaliar as cláusulas where das queries que compõem um caminho, como descrito na subseção 4.7.3;
- obter os valores de atributos pertencentes ao conjunto $\text{C}_{not\_all\_rel}$ na execução de uma query no âmbito do atravessamento de um caminho, como descrito na subseção 4.7.3;

- obter os valores de atributos inquiridos por uma query para verificar se a cláusula where da mesma está associada ao contexto corrente, como descrito na subseção 4.7.4;

- obter os valores de atributos inquiridos por uma query para verificar se através da respetiva query o utilizador percebe que o valor de um atributo sensível de predicado é inválido no contexto corrente, como descrito na subseção 4.7.7.

De acordo com esta descrição é fácil perceber que é possível entrar num ciclo infinito. Para evitar que tal aconteça, a solução encontrada passou por criar um conjunto ($C_{already\_searching}$) que armazena os nomes dos atributos pelos quais se está já a procurar. Posteriormente, quando se pretende obter recursivamente valores de *novos* atributos, remove-se os contidos em $C_{already\_searching}$.

Para o caso de uso apresentado com base nas queries presentes na listagem 4.23 o utilizador pode obter o valor de UnitPrice a partir do tuplo sensível, entre outros, através do caminho $C_1 = \{ q_1, q_2, q_4, q_5\}$.

```
1 QILL: select CustomerID, CompanyName, ContactName from Customers join Orders on
      Customers.CustomerID = Orders.CustomerID join OrderDetails on Orders.
     OrderID = OrderDetails.OrderID and UnitPrice > 50.00;
2
3 Q1: select CustomerID, CompanyName, ContactName from Customers;
4 Q2: select CompanyName, OrderID from Orders join Customers on Orders.CustomerID
      = Customers.CustomerID;
5 Q3: select CustomerID, OrderID from Orders where OrderID < 100;
6 Q4: select OrderID, EmployeeID OrderDate, RequiredDate from OrderDetails;
7 Q5: select OrderDate, ProductID, UnitPrice from Orders join OrderDetails on
     Orders.OrderID = OrderDetails.OrderID;
8 Q6: select RequiredDate, ProductID, ShippedDate, UnitPrice from Orders join
     OrderDetails on Orders.OrderID = OrderDetails.OrderID
9 Q7: select ShippedDate, EmployeeID, ShipVia from Orders;
```

Listagem 4.23: Definição de caso de uso no âmbito do problema da recursividade

Quando se começa a percorrer o caminho $C_{already\_searching} = \{$ UnitPrice$\}$. No âmbito da execução de q_2 percebe-se que o conjunto de atributos cujos valores obtidos não estarão todos associados ao tuplo sensível é composto por $C_{not\_all\_rel} = \{$ OrderID$\}$. Neste contexto, tenta-se obter os valores de OrderID válidos no âmbito do tuplo sensível através dos caminhos $C_{1.1} = \{ q_2\}$ e $C_{1.2} = \{ q_3\}$. Nesta situação, para percorrer $C_{1.1}$ e $C_{1.2}$, não é necessário efetuar nenhuma pesquisa recursiva.

Posteriormente, no âmbito da execução de q_5, $C_{not\_all\_rel} = \{$ ProductID, UnitPrice$\}$. Neste caso, para todas as especificações do contexto do tuplo sensível, compostas pelos valores dos atributos { OrderID, EmployeeID OrderDate, RequiredDate}, procurar-se-á por $C_{not\_all\_rel}$ - $C_{already\_searching} = \{$ ProductID$\}$. A procura de valores de ProductID associados ao contexto corrente pode ser executada recorrendo aos dois caminhos $C_{1.3} = \{ q_5\}$ e $C_{1.4} = \{ q_6\}$. No contexto de $C_{1.3}$ não é necessário efetuar nenhuma chamada recursiva, visto que $C_{not\_all\_rel} = \{$ ProductID, UnitPrice$\}$ mas $C_{already\_searching} = \{$ ProductID, UnitPrice$\}$; ou seja, $C_{not\_all\_rel}$ - $C_{already\_searching} = \varnothing$.

No entanto, ao executar q_6, no âmbito de $C_{1.4}$, $C_{not\_all\_rel} = \{$ ProductID, ShippedDate$\}$ e $C_{already\_searching} = \{$ ProductID, UnitPrice$\}$. Isto faz com que seja necessário procurar

recursivamente pelo atributo ShippedDate. No âmbito desta segunda chamada recursiva $C_{already\_searching}$ = { ProductID, UnitPrice, ShippedDate}, sendo que se pode obter os valores de ShippedDate através dos caminhos $C_{1.4.1}$ = { q_6} e $C_{1.4.2}$ = { q_7}. Ao percorrer $C_{1.4.1}$ não é necessário efetuar nenhuma chamada recursiva, uma vez que, no âmbito de q_6, $C_{not\_all\_rel}$ - $C_{already\_searching}$ = ∅.

Porém, para obter os valores de ShippedDate através de $C_{1.4.2}$ é preciso procurar os valores de ShipVia uma vez que ShipVia ∈ $C_{not\_all\_rel}$ de q_7 e ShipVia ∉ $C_{already\_searching}$. Por fim, no âmbito desta última chamada recursiva pode-se obter ShipVia através de $C_{1.4.2.1}$ = { q_7}, não sendo necessária mais nenhuma chamada recursiva, uma vez que para q_7, $C_{not\_all\_rel}$ ∈ $C_{already\_searching}$.

Neste exemplo pôde-se perceber como esta solução evita que se entre num ciclo infinito na procura pelos atributos pertencentes a $C_{not\_all\_rel}$ de cada query que compõe cada caminho. É também fácil compreender que este comportamento faz com que quantas mais chamadas recursivas forem efetuadas menos exatos são os resultados obtidos em cada uma dessas chamadas. No entanto esta perda de exatidão, à medida que mais chamadas recursivas são executadas, verifica-se também quando o utilizador efetua este processo.

4.7.10 Descrição do relatório produzido

Nesta subseção será descrito o relatório disponibilizado a um administrador de sistema que pretenda utilizar a plataforma desenvolvida no âmbito desta dissertação.

Tal como referido no capítulo 1, o objetivo da plataforma desenvolvida no âmbito desta dissertação passa por avaliar o modelo de controlo de acesso implementado. Mediante o resultado desta avaliação o administrador do sistema pode então manter/alterar o respetivo conjunto de operações autorizadas. Devido à natureza deste problema, uma resposta do tipo protegido/não protegido, no caso de se detetar que existe forma de violar as políticas de controlo de acesso estabelecidas, pode não ser suficiente para o administrador perceber devido a que operações é que se dá a quebra das políticas estabelecidas. Tendo isto em conta, sentiu-se a necessidade de ao nível do relatório produzido, incluir informação que permita a um administrador perceber através de que operações é possível quebrar as políticas de controlo de acesso definidas.

Um excerto de um relatório criado no âmbito da execução do processo de avaliação é apresentado na figura F.1 do Anexo F. Este relatório foi efetuado para um caso de uso composto pelas queries apresentadas na listagem 4.24.

```
1 QILL: select CustomerID, CompanyName, ContactName from Customers join Orders on
      Customers.CustomerID = Orders.CustomerID join OrderDetails on Orders.
    OrderID = OrderDetails.OrderID and UnitPrice > 250.00;
2
3 Q1: select CustomerID, CompanyName, ContactName from Customers;
4 Q2: select ContactTitle, CustomerID, City from Customers
5 Q3: select City, OrderID, OrderDate, RequiredDate, EmployeeID from Orders join
    Customers on Orders.CustomerID = Customers.CustomerID;
6 Q4: select UnitPrice, RequiredDate from OrderDetails join Orders on
    OrderDetails.OrderID = Orders.OrderID where EmployeeID <= 6;
```

Listagem 4.24: Definição de caso de uso no âmbito da apresentação do relatório produzido

Através da análise ao mesmo, pode-se constatar que se tenta obter valores de UnitPrice através do caminho $C_1 = \{ q_2, q_3, q_4\}$. No momento em que se tenta atravessar q_3, obtém-se um conjunto de atributos cujos valores obtidos podem não estar todos associados ao tuplo sensível: $C_{not\_all\_rel} = \{$ OrderID, OrderDate, RequiredDate, EmployeeID$\}$. Neste contexto, através de recursividade tenta-se obter os possíveis valores válidos dos respetivos atributos. No âmbito da formatação do relatório, esta chamada recursiva é representada através de uma deslocação para a direita na respetiva justificação. Pode-se verificar que após a obtenção dos valores válidos dos atributos pertencentes a $C_{not\_all\_rel}$ a justificação do formato do relatório é novamente deslocada para a esquerda. Este comportamento torna o relatório mais percetível, tornando a sua análise mais simples.

Pode-se verificar que após o atravessamento do caminho, as operações realizadas correspondem às apresentadas no âmbito do algoritmo 5. Primeiramente tenta-se filtrar possíveis valores do atributo sensível de predicado UnitPrice através de cláusulas where (tal como descrito na subseção 4.7.4). No entanto, tal como apresentado na listagem 4.24, como não são executadas queries com o atributo UnitPrice presente no respetivo predicado, aborta-se este processamento. Por fim, (1) são removidos conjuntos de resultados sobrepostos (como apenas é atravessado um caminho, este problema não se coloca), (2) são removidos possíveis valores inválidos, de acordo com o processamento descrito em 4.7.7 (sendo que, também ao nível deste processamento, não são removidos valores, uma vez que não são executadas queries que permitam efetuá-lo) e (3) são normalizados os resultados obtidos, tal como descrito em 4.7.8.

Em conjunto com a descrição da forma como a plataforma desenvolvida funciona e as queries utilizadas em cada momento, são disponibilizadas também no relatório as coleções que compõem os resultados obtidos, sobre os quais a avaliação é efetuada. A título de exemplo, a figura F.2 apresentada no Anexo `F`, corresponde a um excerto de uma coleção que armazena os resultados obtidos no âmbito do processo descrito ao longo da presente seção.

4.7.11 Avaliação dos Resultados Obtidos

A forma como se avaliam dos resultados obtidos através

- dos múltiplos caminhos;
- da informação inferida a partir das cláusulas where das queries submetidas pelo utilizador.

será descrita nesta subseção.

Tal como referido na subseção `Avaliação dos resultados obtidos` 3.3.2, quando se avalia uma coleção do tipo REL, basta que apenas uma especificação do contexto corrente seja válida no âmbito de uma cláusula where para considerar que o respetivo contexto é válido no âmbito da mesma.

Por outro lado, caso os resultados obtidos através dos caminhos sejam do tipo NAR, para considerar que o contexto corrente é válido no âmbito da cláusula where é necessário que todas as especificações presentes na respetiva coleção sejam válidas.

Este conceito é estendido aos múltiplos níveis em que os resultados obtidos podem estar organizados. Ou seja, caso os resultados estejam armazenados numa árvore de coleções cuja

representação da topologia é apresentada na figura 4.14, o contexto em que a coleção identificada por `Col1_REL` é criada é válido no âmbito de uma cláusula where (w_1) caso a segunda linha da mesma seja válida. Se tal não se verificar, apenas se pode considerar que o contexto corrente supramencionado é válido em w_1 se as especificações obtidas em `Col3_NAR_Col2` e a segunda linha de `Col2_NAR_Col1` forem válidas.

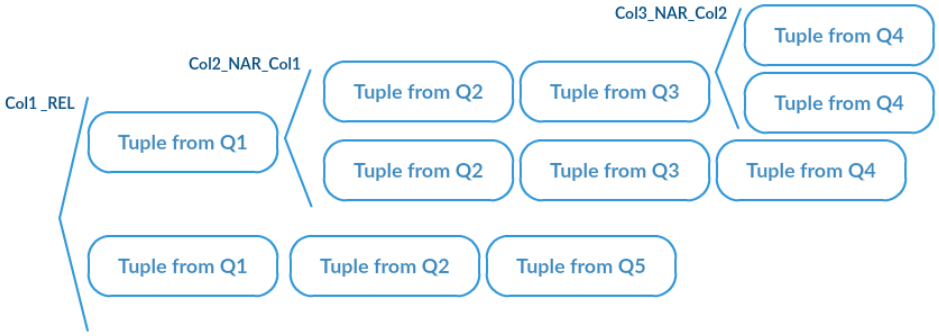

Figura 4.14: Representação da organização dos resultados obtidos

Adicionalmente, para o caso dos resultados obtidos serem compostos apenas por valores inválidos no âmbito de uma cláusula where, apenas se pode considerar que um dado contexto corrente é inválido no âmbito da respetiva cláusula where caso se saiba que foram inquiridos todos os possíveis valores dos atributos sensíveis de predicado.

Para tal, recorreu-se a uma abordagem semelhante à apresentada na subseção `Remoção de resultados sobrepostos` 4.7.6. Neste caso recorreu-se à contagem do número de tuplos válidos no contexto corrente passíveis de serem obtidos nas especificações dadas pelos caminhos presentes nas coleções que contêm os resultados obtidos. Estas especificações do contexto corrente são dadas pelas cláusulas where das queries que compõem cada caminho.

A título de exemplo, para as queries apresentadas na listagem 4.25, suponha-se que para um tuplo sensível identificado por (CustomerID = 'cid1') se pode obter os possíveis valores de UnitPrice associados ao mesmo através dos caminhos

- $C_1 = \{q_1, q_2, q_3\}$;
- $C_2 = \{q_1, q_4\}$;
- $C_3 = \{q_1, q_2, q_5\}$.

Neste caso os contextos dos respetivos caminhos são definidos por

- Ctx_{C1} = (OrderID > 100 and ProductID > 5);
- Ctx_{C2} = (OrderDate > '1996-05-03');
- Ctx_{C3} = (OrderID > 100).

```
1 QILL: select CustomerID, CompanyName, ContactName from Customers join Orders on
      Customers.CustomerID = Orders.CustomerID join OrderDetails on Orders.
    OrderID = OrderDetails.OrderID and UnitPrice > 50.00;
2
3 Q1: select CustomerID, CompanyName, ContactName from Customers;
```

```
4 Q2: select CustomerID, OrderID, EmployeeID, OrderDate, RequiredDate from Orders
      where OrderID > 100;
5 Q3: select OrderID, UnitPrice from OrderDetails where ProductID > 5;
6 Q4: select CustomerID, UnitPrice from Orders join OrderDetails on Orders.
      OrderID where OrderDate > '1996-05-03';
7 Q5: select OrderDate, RequiredDate, UnitPrice from Orders join OrderDetails on
      Orders.OrderID;
```

Listagem 4.25: Definição de caso de uso no âmbito da avaliação dos resultados obtidos processados

Se todos os possíveis valores obtidos de UnitPrice forem inválidos no âmbito da cláusula where de q_{ill}, apenas se pode considerar que o respetivo tuplo corresponde a um tuplo sensível não protegido caso o número de tuplos válidos no âmbito do contexto do tuplo sensível e inválidos no âmbito de Ctx_{C1}, Ctx_{C2} e Ctx_{C3} seja zero. Este contexto pode ser traduzido por (CustomerID = 'cid1' and (OrderID <= 100 or ProductID <= 5) and OrderDate <= '1996-05-03' and OrderID <= 100).

4.8 Utilização da plataforma

Tal como referido neste capítulo, o componente `Orchestrator` corresponde ao módulo que serve de interface de utilização da plataforma desenvolvida.

Na figura 4.15 é apresentado um caso de uso da interação necessária para utilizar esta plataforma. Pode-se constatar que através deste módulo é possível então, após a inicialização da base de dados auxiliar, submeter o caso de uso que se pretende executar.

```
String stm1 = "select `CustomerID`, `CompanyName`, `ContactName` from `Customers`";
String qill1 = "select `Customers`.`CustomerID`, `CompanyName`, `ContactName` from `Customers` join "
    + "`Orders` on `Customers`.`CustomerID` = `Orders`.`CustomerID` join  `order details` on "
    + "`Orders`.`OrderID` = `order details`.`OrderID` where `order details`.`UnitPrice` > 250.0";
Orchestrator orchestrator = new Orchestrator(dburl, username, pass, hypergraphPath, itr);
orchestrator.initializeHypergraphDatabase();
orchestrator.setQueryIllegal(qill1, 0.5);
orchestrator.executeQuery(stm1, false);
orchestrator.executeQuery(stm2, false);
orchestrator.executeQuery(stm3, false);
orchestrator.executeQuery(stm4, false);
orchestrator.executeQuery(stm5, true);
orchestrator.close();
```

Figura 4.15: Utilização da plataforma PFPT

Através da flag `needToEval` definida no âmbito do método `executeQuery`, apresentado na figura 4.16, pode-se também definir a forma como o caso de uso que é submetido é avaliado, na medida em que se fornece flexibilidade ao nível dos momentos em que este processo é executado.

Adicionalmente, é possível definir os atributos que não se pretendem processar através do método `setAttributeAsNonProccessable` tal como apresentado na figura 4.17. Estes atributos devem ser definidos após a inicialização do módulo Orchestrator e antes da inicialização da base de dados de grafos.

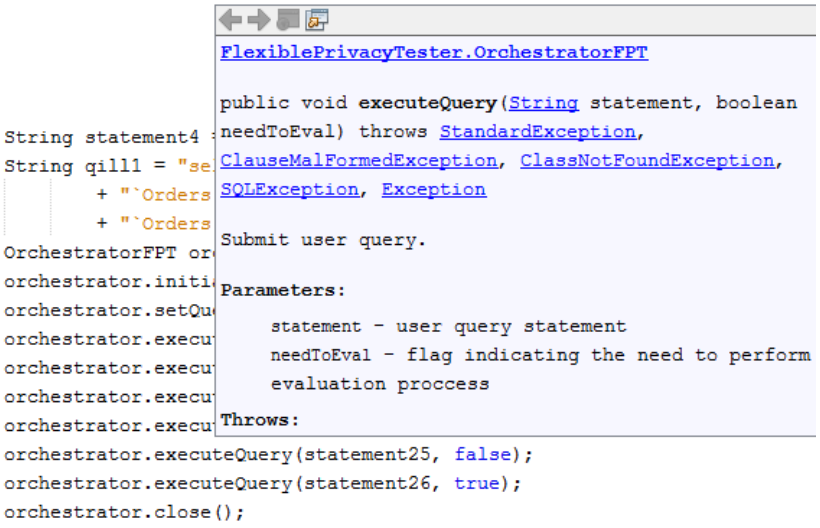

Figura 4.16: Definição do método executeQuery

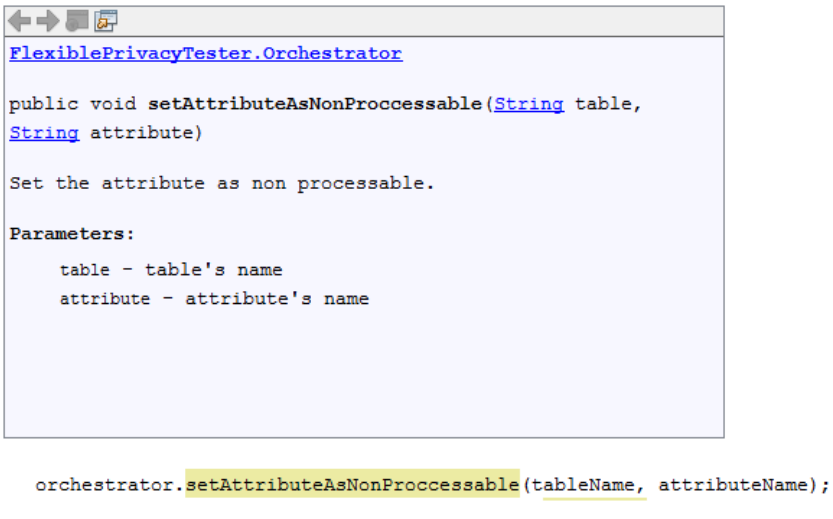

Figura 4.17: Definição do método setAttributeAsNonProccessable

4.9 Performance do Avaliador (caso de uso dinâmico)

O caso de uso no qual se baseia a demonstração da solução apresentada assenta numa instância da base de dados cujo esquema é apresentado na figura A.1. A base de dados relacional utilizada foi o MySQL, armazenada na mesma máquina onde foi executado o caso de teste: Windows 8.1 de 64 bits, Intel i7-4510 @ 2.00GHz, 8GB de RAM e HDD.

Os tempos de execução apresentados foram obtidos recorrendo à chamada do sistema `System.nanoTime()`, que utiliza o relógio de alta definição da JVM e retorna o respetivo valor com uma precisão de nanossegundos.

A listagem 4.26 contém as queries que compõem o caso de uso discutido no âmbito desta seção.

```
1 QILL: select CustomerID, CompanyName, ContactName from Customers join Orders on
      Customers.CustomerID = Orders.CustomerID join OrderDetails on Orders.
```

```
    OrderID = OrderDetails.OrderID and UnitPrice > 250.00;
2
3 Q1: select CustomerID, CompanyName, ContactName from Customers;
4 Q2: select OrderID, ContactTitle from Orders join Customers on Orders.
    CustomerID = Customers.CustomerID;
5 Q3: select City, OrderID, OrderDate, RequiredDate, EmployeeID from Orders join
    Customers on Orders.CustomerID = Customers.CustomerID;
6 Q4: select Country, ContactName from Customers;
7 Q5: select ContactTitle, CustomerID, City from Customers;
8 Q6: select UnitPrice, RequiredDate from OrderDetails join Orders on
    OrderDetails.OrderID = Orders.OrderID where EmployeeID <= 6;
9 Q7: select OrderDate, UnitPrice from OrderDetails join Orders on OrderDetails.
    OrderID = Orders.OrderID where EmployeeID > 3;
10 Q8: select Country, UnitPrice from Orders join Customers on Orders.CustomerID =
     Customers.CustomerID join OrderDetails on OrderDetails.OrderID = Orders.
    OrderID where Country <> 'Brazil' and Country <> 'USA';
11 Q9: select OrderID, ShipVia from Orders;
12 Q10: select OrderDate, ShippedDate, UnitPrice from OrderDetails join Orders on
    OrderDetails.OrderID = OrderID.OrderID where ShipVia > 1;
```

Listagem 4.26: Definição do caso de uso

De seguida, em 4.9.1, abordar-se-á a evolução do tempo requerido pelo processo de avaliação e do número de tuplos efetivamente avaliados, à medida que o caso de uso cresce.

Na subseção 4.9.2, comparar-se-ão os resultados obtidos tanto em termos de tempo de execução como de tuplos sensíveis efetivamente avaliados recorrendo à estratégia de flexibilidade na utilização de atributos no processo de avaliação.

4.9.1 Performance do avaliador com a evolução do caso de uso

Antes efetuar uma análise aos resultados obtidos para o caso de uso supramencionado, convém mencionar que a mesma assenta num cenário que tipicamente a solução desenvolvida não permite aconteder. Como se poderá perceber ao longo desta subseção, o limite da probabilidade de acerto em tuplos sensíveis protegidos será ultrapassado chegando mesmo a ser possível, através das operações autorizadas , inferir quais são os tuplos sensíveis protegidos. Esta situação é facilmente detetada pela plataforma, no entanto, para caso de teste e para melhor poder explicar o modo de funcionamento da mesma optou-se por esta abordagem.

A evolução da performance, à medida que o caso de uso descrito na listagem 4.26 é formado, é apresentada na tabela 4.4.

Relativamente à evolução do caso de uso, pode-se perceber que apenas após a submissão de q_6 se pode obter possíveis valores de UnitPrice, através de $C_1 = \{ q_5, q_3, q_6\}$.

Posto isto, analisando o gráfico apresentado na figura 4.18 pode-se constatar que os tempos de avaliação quando o número de queries é menor do que seis são aproximadamente nulos.

No âmbito do processo de avaliação aquando da execução de q_6 pode-se perceber que o mesmo passa a ter um valor não nulo. Isto deve-se ao atravessamento de C_1. Neste contexto, após se obterem os valores de { ContactTitle, City} através de q_5, quando se atravessa q_3, é efetuada uma chamada recursiva para obter os possíveis valores válidos dos atributos de $C_{not\_all\_ref}$ = { OrderID, OrderDate, RequiredDate, EmployeeID}. Estes possíveis valores válidos são obtidos, tipicamente, através da interseção dos resultados obtidos através de $C_{1.1} = \{ q_3\}$ e $C_{1.2} = \{ q_2\}$.

#Queries	Test#	Tempo (segundos)	#Válidos	#Inválidos	#Desconhecidos
1	1	1.674	0	0	93
	2	0.598			
	3	0.597			
2	1	0.628	0	0	93
	2	0.135			
	3	0.175			
3	1	0.499	0	0	93
	2	0.146			
	3	0.187			
4	1	0.632	0	0	93
	2	0.33			
	3	0.122			
5	1	0.207	0	0	93
	2	0.121			
	3	0.255			
6	1	222.281	0	16	77
	2	227.265			
	3	218.978			
7	1	216.758	2	68	23
	2	221.212			
	3	223.145			
8	1	205.033	6	75	12
	2	193.282			
	3	203.557			
9	1	248.638	6	75	12
	2	257.671			
	3	261.731			
10	1	1087.246	7	76	10
	2	1090.491			
	3	1133.731			

Tabela 4.4: Evolução da performance do avaliador

Através de C_1 é possível perceber que 16 dos 93 customers presentes na base de dados correspondem a tuplos sensíveis não protegidos. Esta perceção deve-se:

- ao fato de os valores de RequiredDate aos quais os respetivos customers estão associados, não estarem associados a nenhum valor de (UnitPrice > 250);
- estarem apenas associados a Orders cujos (EmployeeID <= 6).

Uma amostra de uma coleção formada no âmbito deste caminho é apresentada na figura F.3, apresentada no Anexo F. Convém mencionar que os *QueryID* começam a partir de 0 e, portanto, são inferiores em uma unidade aos mencionados na listagem 4.26.

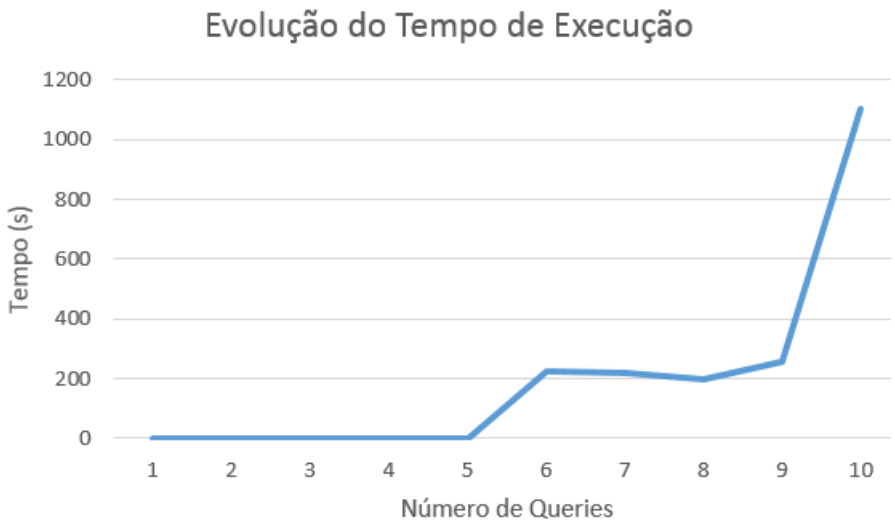

Figura 4.18: Evolução do tempo de avaliação

Após a execução de q_7 é possível obter possíveis valores de UnitPrice através também de C_2 = { q_5, q_3, q_7}. Neste contexto, pode-se verificar que o tempo no processo de avaliação se mantém. Este comportamento deve-se a uma otimização implementada. Na realidade o processo descrito no algoritmo 5, não corresponde efetivamente à implementação efetuada. De fato, no âmbito de cada iteração do ciclo `while` é atravessado um caminho e obtidos os respetivos resultados. Posteriormente, após a operação de agregação de múltiplos resultados, a coleção resultante é avaliada no âmbito da cláusula where de q_{ill}. Se nesse momento for possível ter acesso a um resultado conclusivo da avaliação (válido/inválido) o processamento é *interrompido*. Neste contexto as operações efetuadas após o ciclo `while`, cujo tempo de execução não pode ser desprezado, não são também executadas. Posto isto, o fato de o número de tuplos efetivamente avaliados aumentar, o que em alguns casos se traduz no fato de o processo supramencionado ser interrompido, atenua de certa forma o caminho adicional atravessado.

No âmbito do processo de avaliação do caso de uso composto pelas primeiras 7 queries, os 16 tuplos sensíveis supramencionados continuam a necessitar apenas de percorrer C_1 para perceber que correspondem a tuplos sensíveis não protegidos.

Adicionalmente, através do atravessamento de C_2 e respetiva agregação com C_1 (para orders com (3 < OrderID <= 6)), pode-se perceber que se fica a conhecer um customer que está associado a um valor de (UnitPrice > 250) e outros 51 customers que não estão de fato associados a um valor de (UnitPrice > 250). A figura F.4 apresentada no Anexo `F` mostra um excerto da coleção através da qual é possível perceber que o customer identificador por (CustomerID = 'BERGS') corresponde a um tuplo sensível protegido. Esta associação é possível devido ao fato de, no âmbito do atravessamento de C_2, através de q_3 e q_2 se conseguir obter tuplos de q_3, os quais se sabe estarem efetivamente associados ao customer supramencionado.

Após a execução de q_8 é possível obter os valores de UnitPrice através do caminho C_3= { q_4, q_8}. Como a maioria dos valores de Country estão associados apenas a valores de UnitPrice inválidos no âmbito da cláusula where de q_{ill}, é possível através dos resultados obtidos neste caminho perceber muitos tuplos sensíveis como não protegidos. Como este caminho é o

primeiro a ser atravessado (por ser composto apenas por duas queries) e pela mesma razão previamente apresentada, é possível em muitos dos tuplos sensíveis avaliados interromper o processamento descrito na seção 4.7 e assim efetuar o processo de avaliação em menos tempo.

Relativamente ao processo de avaliação após a execução de q_9, não é formado nenhum caminho adicional através do qual seja possível obter valores de UnitPrice. No entanto, no âmbito do atravessamento de C_1 e C_2, esta query é também utilizada no âmbito da obtenção de possíveis valores de OrderID válidos, apesar de a mesma não produzir resultados concretos em termos de tuplos efetivamente avaliados.

Através da execução de q_{10}, é possível obter possíveis valores de UnitPrice através de C_4 = { q_5, q_3, q_{10}}. Neste contexto, pode-se verificar que volta a existir um aumento do tempo do processo de avaliação, desta vez significativo, sendo que é também possível descobrir mais um tuplo sensível protegido e outro não protegido.

4.9.2 Performance do avaliador fornecendo flexibilidade na utilização de atributos

Tal como referido, esta solução tende a ser bastante exigente computacionalmente. Para fornecer alguma flexibilidade, foi desenvolvida a estratégia apresentada na subseção `Flexibilidade na utilização de atributos no processo de avaliação` 3.1.2.

No âmbito do caso de uso apresentado na listagem 4.26, optou-se por, num primeiro instante, definir apenas o atributo OrderDate. Perante este cenário as queries q_3, q_7 e q_{10} foram transformadas, passando a inquirir todos os atributos da tabela Orders. Esta transformação é refletida na listagem 4.27. A evolução da performance nesta abordagem é apresentada na tabela 4.5.

```
1 Q3.1: select City, OrderID, OrderDate, RequiredDate, EmployeeID, CustomerID,
    ... from Orders join Customers on Orders.CustomerID = Customers.CustomerID;
2 Q7.1: select OrderDate, UnitPrice, OrderID, CustomerID, ... from OrderDetails
    join Orders on OrderDetails.OrderID = Orders.OrderID where EmployeeID > 3;
3 Q10.1: select OrderDate, ShippedDate, UnitPrice, OrderID, CustomerID ... from
    OrderDetails join Orders on OrderDetails.OrderID = OrderID.OrderID where
    ShipVia > 1;
```

Listagem 4.27: Transformação das queries do caso de uso (1)

À semelhança do caso de uso previamente definido, apenas a partir de q_6 é possível obter os possíveis valores de UnitPrice. Posto isto, após a execução de q_6, esta obtenção pode ser realizada através de $C_{1.1}$ = { $q_{3.1}$, q_6}. Neste contexto, através da comparação das linha azul e linha laranja a tracejado do gráfico presente na figura 4.19 é possível verificar uma descida no tempo necessário para efetuar o processo de avaliação. Esta variação deve-se ao fato de no âmbito de $q_{3.1}$, com a presença do atributo CustomerID no conjunto dos atributos inquiridos, $C_{not\_all\_rel} = \varnothing$. Adicionalmente a solução desenvolvida no âmbito desta dissertação percebe que os mesmos 16 dos 93 customers presentes na base de dados correspondem a tuplos sensíveis não protegidos. Isto deve-se ao fato de para estes customers, já a versão *normal* da solução desenvolvida ter percebido que as respetivas orders estavam efetivamente associadas a eles.

#Queries	Test#	Tempo (segundos)	#Válidos	#Inválidos	#Desconhecidos
1	1	1.745	0	0	93
	2	0.6			
	3	0.114			
2	1	1.605	0	0	93
	2	0.643			
	3	0.195			
3	1	1.151	0	0	93
	2	0.146			
	3	0.213			
4	1	1.086	0	0	93
	2	0.615			
	3	0.18			
5	1	0.653	0	0	93
	2	0.606			
	3	0.109			
6	1	154.209	0	16	77
	2	154.687			
	3	152.918			
7	1	87.866	6	74	13
	2	97.634			
	3	97.549			
8	1	140.036	9	75	11
	2	138.029			
	3	133.383			
9	1	143.846	9	75	11
	2	130.815			
	3	142.946			
10	1	259.940	12	76	5
	2	236.859			
	3	238.474			

Tabela 4.5: Evolução da performance do avaliador

Após a execução de $q_{7.1}$, adicionalmente, pode-se obter os possíveis valores de UnitPrice através de $C_{2.1} = \{ q_{7.1}\}$. Como os resultados obtidos através de $q_{7.1}$ são sempre associados aos respetivos customers, recorrendo também a $C_{1.1}$, consegue-se perceber 6 tuplos como tuplos sensíveis protegidos e 74 como tuplos sensíveis não protegidos, dos 93 existentes na base de dados. Comparando a performance em termos dos tempos obtidos através desta abordagem e da anterior, pode-se também constatar que o tempo do processo de avaliação desta é inferior.

Após a execução de q_8 e q_9, adicionalmente, é possível obter possíveis valores de UnitPrice através de $C_{3.1} = \{ q_4, q_8\}$; sendo que os resultados obtidos neste processo de avaliação seguem a tendência dos anteriores: em menos tempo de execução é possível perceber a natureza (protegido ou não protegido) de mais tuplos sensíveis.

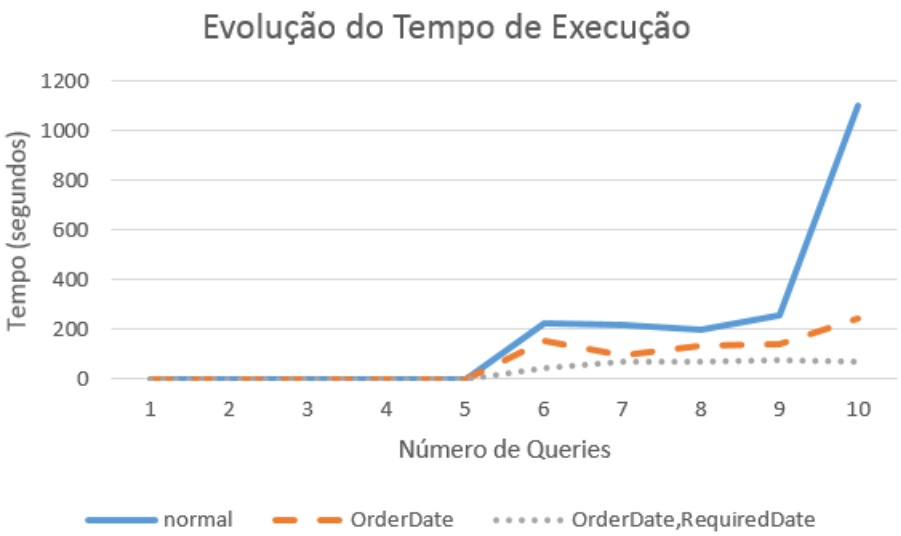

Figura 4.19: Evolução do tempo de avaliação variando os atributos usados

Por fim, no âmbito da execução de $q_{10.1}$, forma-se um novo caminho a partir do qual é possível obter possíveis valores de UnitPrice: $C_{4.1} = \{ q_{10.1}\}$. Através do atravessamento deste caminho único é possível perceber mais 3 tuplos sensíveis como protegidos e 1 como não protegido.

Por último, estendeu-se esta estratégia também ao atributo RequiredDate. Como consequência as queries apresentadas na listagem 4.28 foram transformadas.

```
1 Q3.1: select City, OrderID, OrderDate, RequiredDate, EmployeeID, CustomerID,
    ... from Orders join Customers on Orders.CustomerID = Customers.CustomerID;
2 Q6.2: select UnitPrice, RequiredDate, OrderID, EmployeeID, CustomerID, ... from
     OrderDetails join Orders on OrderDetails.OrderID = Orders.OrderID where
    EmployeeID <= 6;
3 Q7.1: select OrderDate, UnitPrice, OrderID, CustomerID, ... from OrderDetails
    join Orders on OrderDetails.OrderID = Orders.OrderID where EmployeeID > 3;Q8
    : select OrderID, ShipVia from Orders;
4 Q10.1: select OrderDate, ShippedDate, UnitPrice, OrderID, CustomerID ... from
    OrderDetails join Orders on OrderDetails.OrderID = OrderID.OrderID where
    ShipVia > 1;
```

Listagem 4.28: Transformação das queries do caso de uso (2)

Neste caso, após a execução de $q_{6.2}$ podem-se obter os valores de UnitPrice através de $C_{1.2} = \{ q_{6.2}\}$. Os resultados obtidos através desta query são todos associados aos respetivos tuplos sensíveis, visto que a projeção se efetua sobre o atributo CustomerID.

Por fim, no processo de avaliação, após a execução de $q_{7.1}$, é possível aferir a natureza de todos os tuplos sensíveis existentes na base de dados, sendo que a execução de queries adicionais não tem impacto no tempo necessário para efetuar os processos de avaliação subsequentes. Este fato está patente na linha cinzenta de do gráfico apresentado na figura 4.19.

4.9.3 Performance do avaliador com a evolução da dimensão da base de dados

No âmbito desta subseção discutir-se-á a performance da solução desenvolvida no âmbito desta dissertação à medida que a dimensão da base de dados aumenta. Neste contexto, o conjunto de operações autorizadas manter-se-á estático. O caso de uso abordado nesta subseção é apresentado na listagem 4.29.

```
1 Q1: select CustomerID, CompanyName, ContactName from Customers where CustomerID
      = 'customerid1';
2 Q2: select CustomerID, OrderID, ContactName from Orders where CustomerID = '
     customerid1';
3 Q3: select OrderDetails.OrderID, UnitPrice from OrderDetails join Orders on
     OrderDetails.OrderID = Orders.OrderID  where CustomerID = 'customerid1';
```

Listagem 4.29: Definição do caso de uso

A análise nesta seção recairá apenas sobre um tuplo sensível. De forma a analisar o comportamento da solução desenvolvida começar-se-á por efetuar medições quando ele está associado a 5 OrderID, estando cada OrderID associada a 5 produtos, através da tabela OrderDetails. Posto isto, aumentar-se-á o número de OrderID associadas ao respetivo tuplo, mantendo a proporção ao nível do número de associações entre OrderID e ProductID na tabela OrderDetails.

#Orders	Test#	Tempo (segundos)
5	1	1.571
	2	1.713
	3	1.625
25	1	3.009
	2	2.889
	3	2.285
50	1	3.306
	2	3.031
	3	3.302
500	1	12.270
	2	13.917
	3	13.212
1000	1	188.575
	2	180.348
	3	182.038

Tabela 4.6: Evolução da performance do avaliador

Através da análise dos tempos de execução apresentados na tabela, pode-se perceber que para números relativamente pequenos de OrderID associados ao tuplo sensível, o tempo de execução sobe de forma controlada. No entanto, atentando na diferença de tempos entre o número de OrderID associadas igual a 500 e igual a 1000 pode-se constatar que se por um lado a razão entre o número de OrderID associados num caso e no outro é de 2, a razão entre os tempos nos dois casos é de aproximadamente 13.8. As figuras F.5 e F.6, apresentadas no

Anexo F mostram a utilização da memória nos casos em que o tuplo sensível está associado a 500 e 1000 OrderID, respetivamente. Como se pode constatar através da análise destas imagens, o Garbage Collector (GC) no segundo cenário efetua muito mais vezes o processo de *garbage collection* do que no âmbito do primeiro. Estes *gc*'s são representados pelos múltiplos picos do gráfico correspondente à *utilização da heap*.

Pode-se então perceber que à medida que se lida com mais informação, além de o volume de informação necessário processar, aumenta também o número de ocorrências dos *gc*'s, o que tem um impacto muito grande na solução desenvolvida no âmbito desta dissertação.

Capítulo 5

Conclusão

O trabalho desenvolvido no âmbito desta dissertação tinha como objetivo o desenvolvimento de uma plataforma que permitisse avaliar se, através de operações autorizadas, era possível inferir informação sensível definida pelas políticas de controlo de acesso probabilísticas.

O objetivo desta plataforma era então auxiliar o administrador do sistema de informação a definir o conjunto das operações autorizadas, de forma a que através do mesmo não se pudessem violar as políticas de controlo de acesso estabelecidas. Neste contexto foi desenvolvida uma solução, que em caso de violação das políticas estabelecidas fornece ao administrador a probabilidade de acerto nos respetivos tuplos sensíveis protegidos e as queries às quais um utilizador malicioso recorre para obter essa informação, tal como apresentado na seção `Descrição do relatório produzido` 4.7.10.

No âmbito da solução desenvolvida, a qual se pretende aproximar o máximo possível do processo de inferência passível de ser realizado por um utilizador *malicioso*, pôde-se perceber que a mesma não é escalável. Em termos de tempo de execução verificou-se, que à medida que a quantidade de queries que compunha um caso de uso aumentava, o mesmo aumentava também. De fato, tal como referido em 4.9.1, recorrendo a algumas otimizações foi possível, para o caso de uso apresentado, mitigar alguma da perda de performance.

Adicionalmente verificou-se também, no âmbito das seções de avaliação de performance do capítulo 4, que esta solução tende também a necessitar de uma grande quantidade de recursos do sistema computacional. Neste contexto, a utilização de uma estratégia recursiva contribui em grande medida para esta necessidade. Uma forma de mitigar a necessidade de recursos por uma solução semelhante a esta poderia passar por recorrer a uma estratégia iterativa em vez de recursiva. Deste modo, a JVM não necessita de armazenar o contexto de chamadas recursivas.

Relativamente à forma como a informação passível de ser inferida por um utilizador era armazenada, foi necessário o desenvolvimento de coleções. Tal como mencionado no capítulo 4, estas deveriam armazenar a informação sob a forma de árvore e permitir lidar com grandes volumes de informação na medida em que não fosse exigida a manutenção dessa informação em memória. No âmbito da primeira solução desenvolvida, apresentada na seção `Desenvolvimento de coleções suportadas por ficheiros` do Anexo `B`, ambos os requisitos foram alcançados. No entanto, no âmbito da segunda abordagem, apresentada em `Desenvolvimento de coleções suportadas por base de dados de hipergrafos` 4.7.3, não se cumpriu o segundo requisito: não necessitar que toda a informação seja armazenada

em memória. De fato, nas coleções suportadas pela base de dados HyperGraphDB, não é mantida em memória toda a informação contida nas linhas das mesmas mas sim a referência para as respetivas linhas. Apesar de apenas serem armazenadas as referências para as linhas das coleções, pôde-se perceber através da análise da figura F.6 que esta solução não consegue suportar grandes quantidades de informação.

5.1 Trabalho Futuro

Devido à natureza do problema abordado nesta dissertação, apesar de ter sido apresentada uma solução que resolve as duas técnicas de inferência de informação que permitem ludibriar os tradicionais mecanismos de controlo de acesso, é possível que não tenham sido exploradas e todas as formas da mesma. Posto isto, torna-se necessário efetuar um estudo completo sobre possíveis técnicas de inferência e, caso tal seja necessário dotar a plataforma desenvolvida de formas de as suportar.

Adicionalmente, no âmbito desta dissertação foi apenas abordado o paradigma de consulta de informação. No entanto este problema pode existir também ao nível da inserção, atualização e remoção de informação nas bases de dados. Posto isto, para utilizar esta plataforma num sistema de informação em que seja permitido alterar informação na base de dados além de a consultar, torna-se necessário estendê-la para suportar estes casos.

Tal como ficou patente na seção `Interpretação de Queries` 4.4, esta solução possui algumas limitações ao nível das queries suportadas. Queries com os operadores união, interseção, diferença e com funções de agregação não são suportadas, pelo que poderia ser interessante a sua implementação.

A título de exemplo, uma possível solução para as queries que possuam o operador *UNION*, poderia passar por executar cada uma em separado e processá-las como se de queries distintas se tratassem. Assim, no âmbito de uma abordagem baseada em hipergrafos, recorrendo a um nó especial que represente a operação de união, ao associá-lo a cada um dos nós de queries que a compõem era criada uma estrutura hierárquica. Relativamente aos operadores INTERSECT e MINUS era possível ter uma abordagem análoga, sendo que para o caso em particular da operação de interseção, sabia-se que os tuplos que tinham sido retornados ao utilizador corresponderiam aos que se repetissem no resultado das duas queries.

Por fim, em relação ao trabalho proposto inicialmente, a plataforma desenvolvida apresenta uma limitação. Efetivamente só consegue validar soluções que recorrem a mecanismos de controlo de acesso baseado em técnicas de reescrita de queries e/o utilização de vistas para filtragem dos valores obtidos pelas queries submetidas. Isto deve-se ao fato de a solução partir do princípio que toda a informação retornada pelas queries seja válida. Neste contexto, o paradigma utilizado pelas soluções que aplicam controlo de acesso mascarando a informação sensível, como por exemplo a utilização de valores NULL, viola este princípio.

Bibliografia

[1] X. Yang, C. Li, G. Yu e L. Shi, "Xguard: A system for publishing xml documents without information leakage in the presence of data inference", em *21st International Conference on Data Engineering (ICDE'05)*, abr. de 2005, pp. 1124–1125.

[2] P. Samarati e L. Sweeney, "Protecting privacy when disclosing information: K-anonymity and its enforcement through generalization and suppression", SRI International, rel. téc., 1998.

[3] G. Canfora, C. A. Visaggio e V. Paradiso, "A test framework for assessing effectiveness of the data privacy policy's implementation into relational databases", em *Proceedings International Conference on Availability, Reliability and Security, 2009. ARES'09.*, IEEE, 2009, pp. 240–247.

[4] J. Gosling, *The Java language specification*. Addison-Wesley Professional, 2000.

[5] *Netbeans ide.* (Visitado em Novembro, 2016). endereço: `https://netbeans.org`.

[6] *Foundationdb sql parser.* (Visitado em Novembro, 2016). endereço: `https://libraries.io/github/FoundationDB/sql-parser`.

[7] *Sql parse, analyze, transform and format all in one.* (Visitado em Novembro, 2016). endereço: `http://www.sqlparser.com/`.

[8] *Visitor pattern - wikipedia.* (Visitado em Novembro, 2016). endereço: `https://en.wikipedia.org/wiki/Visitor_pattern`.

[9] J. J. Miller, "Graph database applications and concepts with neo4j", em *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, vol. 2324, 2013.

[10] *Titan distributed graph database.* (Visitado em Novembro, 2016). endereço: `http://titan.thinkaurelius.com`.

[11] *Allegrograph semantic graph database.* (Visitado em Novembro, 2016). endereço: `http://franz.com/agraph/allegrograph`.

[12] *Graphbase - the world's most powerfull graph dbms.* (Visitado em Novembro, 2016). endereço: `http://graphbase.net`.

[13] *Orientdb - distributed multi-model and graph database.* (Visitado em Novembro, 2016). endereço: `http://orientdb.com/orientdb`.

[14] *Ontotext graph dbTM.* (Visitado em Novembro, 2016). endereço: `http://ontotext.com/products/graphdb`.

[15] *A distributed, fault-tolerant graph database.* (Visitado em Novembro, 2016). endereço: `https://github.com/twitter/flockdb`.

[16] B. Iordanov, "Hypergraphdb: A generalized graph database", em *International Conference on Web-Age Information Management*, Springer, 2010, pp. 25–36.

[17] *Mysql.* (Visitado em Novembro, 2016). endereço: `https://www.mysql.com`.

[18] E. F. Codd, "A relational model of data for large shared data banks", *Communications of the ACM*, vol. 26, nº 1, pp. 64–69, 1983.

[19] J. L. Gross e T. W. Tucker, *Topological graph theory*. Courier Corporation, 1987.

[20] V. I. Voloshin, *Introduction to graph and hypergraph theory*. Nova Science Publ., 2009.

[21] C. Berge, *Hypergraphs: Combinatorics of finite sets*. Elsevier, 1984, vol. 45.

[22] R. Angles e C. Gutierrez, "Survey of graph database models", *ACM Computing Surveys (CSUR)*, vol. 40, nº 1, p. 1, 2008.

[23] N. L. Biggs, *Discrete mathematics*, 1999.

[24] P. Samarati e S. C. de Vimercati, "Access control: Policies, models, and mechanisms", em *International School on Foundations of Security Analysis and Design*, Springer, 2001, pp. 137–196.

[25] J. Rubart, "Context-based access control", em *Proceedings of the 2005 symposia on Metainformatics*, ACM, 2005, p. 13.

[26] C. A. Ardagna, M. Cremonini, E. Damiani, S. D. C. di Vimercati e P. Samarati, "Supporting location-based conditions in access control policies", em *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, ACM, 2006, pp. 212–222.

[27] A. Almehmadi e K. El-Khatib, "On the possibility of insider threat prevention using intent-based access control (ibac)", *IEEE Systems Journal*, vol. PP, nº 99, pp. 1–12, 2015.

[28] D.-K. Kim, I. Ray, R. France e N. Li, "Modeling role-based access control using parameterized uml models", em *International Conference on Fundamental Approaches to Software Engineering*, Springer, 2004, pp. 180–193.

[29] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn e R. Chandramouli, "Proposed nist standard for role-based access control", *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, nº 3, pp. 224–274, 2001.

[30] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone et al., "Guide to attribute based access control (abac) definition and considerations (draft)", *NIST Special Publication*, vol. 800, nº 162, 2013.

[31] V. C. Hu, D. R. Kuhn e D. F. Ferraiolo, "Attribute-based access control.", *IEEE Computer*, vol. 48, nº 2, pp. 85–88, 2015.

[32] S. Rizvi, A. Mendelzon, S. Sudarshan e P. Roy, "Extending query rewriting techniques for fine-grained access control", em *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, ACM, 2004, pp. 551–562.

[33] R. Halder e A. Cortesi, "Observation-based fine grained access control for relational databases.", em *ICSOFT (1)*, 2010, pp. 254–265.

[34] N. Sehta e S. Jain, "A fine grained access control model for relational databases", *IJCSIT*, vol. 3, nº 1, pp. 3183–3186, 2012.

[35] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl et al., "System r: Relational approach to database management", *ACM Transactions on Database Systems (TODS)*, vol. 1, nº 2, pp. 97–137, 1976.

[36] J.-W. Byun e N. Li, "Purpose based access control for privacy protection in relational database systems", *The VLDB Journal*, vol. 17, nº 4, pp. 603–619, 2008.

[37] S. Rizvi, A. Mendelzon, S. Sudarshan e P. Roy, "Extending query rewriting techniques for fine-grained access control", em *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, ACM, 2004, pp. 551–562.

[38] Q. Wang, T. Yu, N. Li, J. Lobo, E. Bertino, K. Irwin e J.-W. Byun, "On the correctness criteria of fine-grained access control in relational databases", em *Proceedings of the 33rd international conference on Very large data bases*, VLDB Endowment, 2007, pp. 555–566.

[39] D. E. Denning, S. G. Akl, M. Morgenstern, P. G. Neumann, R. R. Schell e M. Heckman, "Views for multilevel database security", em *1986 IEEE Symposium on Security and Privacy*, abr. de 1986, pp. 156–156.

[40] S. Spendolini, "Virtual private database", em *Expert Oracle Application Express Security*, Springer, 2013, pp. 211–223.

[41] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu e D. DeWitt, "Limiting disclosure in hippocratic databases", em *Proceedings of the Thirtieth international conference on Very large data bases*, VLDB Endowment, vol. 30, 2004, pp. 108–119.

[42] R. Agrawal, J. Kiernan, R. Srikant e Y. Xu, "Hippocratic databases", em *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB Endowment, 2002, pp. 143–154.

[43] D. D. Regateiro, *A secure, distributed and dynamic rbac for relational applications*, Dissertação de Mestrado, Universidade de Aveiro, 2014.

[44] Ó. M. Pereira, R. L. Aguiar e M. Y. Santos, "Acada: Access control-driven architecture with dynamic adaptation", em *Proceedings of the 2012 International Conference on Software Engineering and Knowledge Engineering (SEKE'2012)*, 2012, pp. 387–393.

[45] O. M. Pereira, D. D. Regateiro e R. L. Aguiar, "Distributed and typed role-based access control mechanisms driven by crud expression", *International Journal of Computer Science Theory and Application*, vol. 2, nº 1, pp. 1–11, 2014.

[46] Ó. M. Pereira, D. D. Regateiro e R. L. Aguiar, "Role-based access control mechanisms", em *2014 IEEE Symposium on Computers and Communications (ISCC)*, jun. de 2014, pp. 1–7.

[47] S. S. Surajit Chaudhuri Tanmoy Dutta, "Fine grained authorization through predicated grants", em *IEEE International Conference on Data Engineering*, IEEE, 2007.

[48] A. Roichman e E. Gudes, "Fine-grained access control to web databases", em *Proceedings of the 12th ACM symposium on Access control models and technologies*, ACM, 2007, pp. 31–40.

[49] *Sql injection.* (Visitado em Novembro, 2016). endereço: `https://www.owasp.org/index.php/SQL_Injection`.

[50] L. Caires, J. A. Pérez, J. C. Seco, H. T. Vieira e L. Ferrão, "Type-based access control in data-centric systems", em *European Symposium on Programming*, Springer, 2011, pp. 136–155.

[51] *Java platform enterprise edition (java ee).* (Visitado em Novembro, 2016). endereço: `http://www.oracle.com/technetwork/java/javaee/overview/index.html`.

[52] *Using java ee annotations and dependency injection.* (Visitado em Novembro, 2016). endereço: `https://docs.oracle.com/cd/E13222_01/wls/docs100/programming/annotate_dependency.html`.

[53] *Specifying authorized users by declaring security roles.* (Visitado em Novembro, 2016). endereço: `https://docs.oracle.com/cd/E19798-01/821-1841/gjgcq/index.html`.

[54] *The java language environment.* (Visitado em Novembro, 2016). endereço: `http://www.oracle.com/technetwork/java/intro-141325.html`.

[55] G. Hamilton, R. Cattell e M. Fisher, *Jdbc Database Access With Java: A Tutorial and Annotated Reference (Java Series)*. Addison-Wesley, ago. de 1997, vol. 7, pp. 5–9.

[56] G. Reese, *Database Programming with JDBC & Java: Developing Multi-Tier Applications (Java (O'Reilly))*. O'Reilly Media, set. de 2000, pp. 41–42.

[57] *Jdbc tutorial.* (Visitado em Novembro, 2016). endereço: `https://www.tutorialspoint.com/jdbc/jdbc_tutorial.pdf`.

[58] F. Bouillé, *The Hypergraph-Based Data Structure: A New Approach to Data Base Modelling and Application*. Springer, 1977, pp. 37–55.

[59] P. E. d. Silveira, R. A. Pereira e U. dos Açores, *Metodologia universal de estruturação dos conhecimentos :* [Ponta Delgada : s. n.], 1983, Trabalho de síntese para apresentação de provas de Aptidão Pedagógica e Capacidade Científica à Universidade dos Açores, sob a orientação de Agonia Pereira.

[60] J. E. Whitesitt, *Boolean algebra and its applications*. Courier Corporation, 1995.

[61] F. Hausdorff, "Set theory", *American Mathematical Society, Providence*, vol. 119, 1957.

Anexo A

Esquema da base de dados *Northwind*

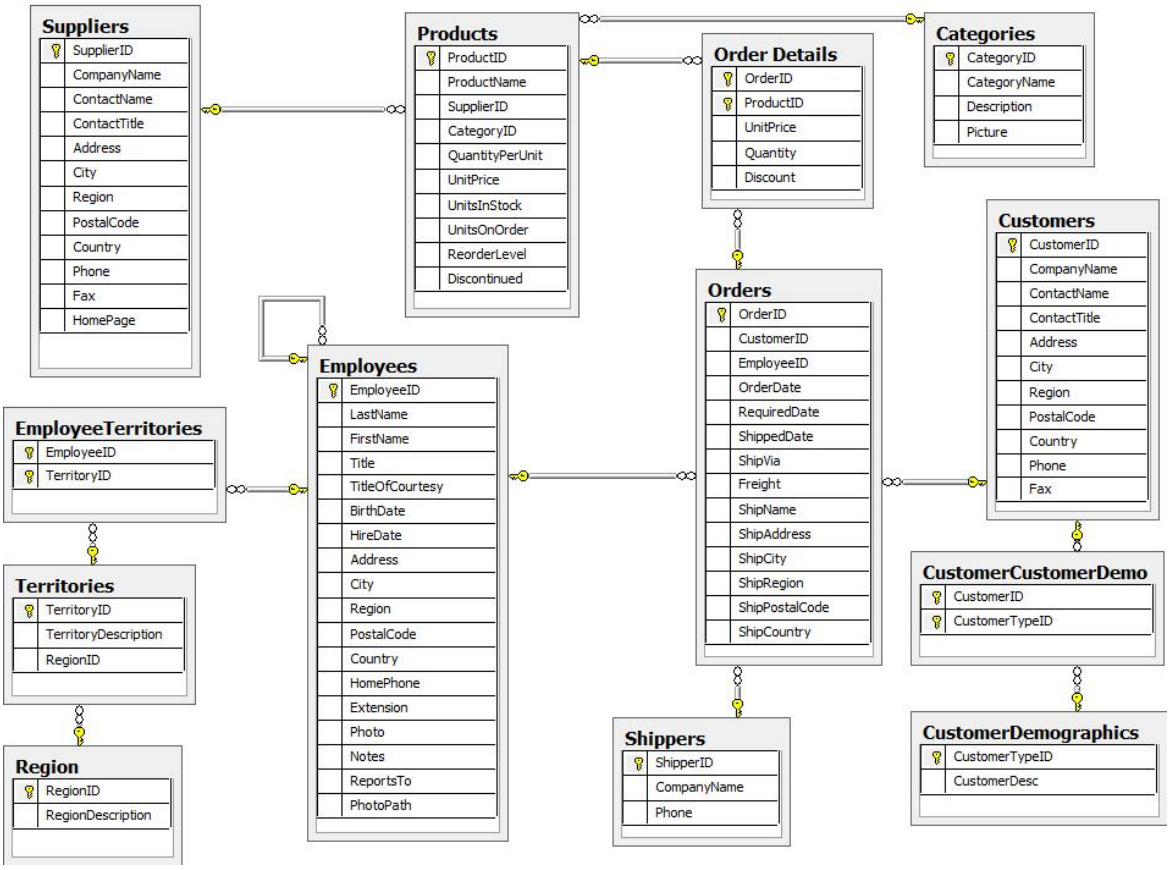

Figura A.1: Esquema da base de dados Northwind

Anexo B

Solução de armazenamento de informação recorrendo a grafos

A solução recorrendo a grafos correspondeu à primeira abordagem ao problema do armazenamento de informação obtida das queries executas. Tal como ficou patente no exemplo apresentado na seção 3.2, a associação entre o empregado com {EmployeeID = 1} e o salário {Salary = 2954.55} é simbolizada pela relação existente no grafo entre os dois nós correspondentes. Este tipo de operação, que na Teoria de Grafos corresponde a encontrar os nós que compõem a vizinhança de um determinado nó, tem complexidade O(1), ou seja, não depende da dimensão do grafo.

Tal como mencionado na seção 3.2, ao passar a informação organizada sob a forma de modelo relacional para a forma de modelo de grafos, os atributos que correspondem a chaves estrangeiras devem ser armazenados com a identificação correspondente ao nome da tabela e do atributo originais. Isto permite que se consiga associar os valores de atributos de múltiplas tabelas corretamente. Para o esquema da base de dados *Northwind* A.1, se um utilizador executar as queries

```
1 q1: select EmployeeID, LastName, FirstName from Employees where EmployeeID = 1;
2 q2: select OrderID, EmployeeID from Orders where OrderID = 5;
```

a representação no grafo dos resultados retornados corresponde à apresentada na figura B.1.

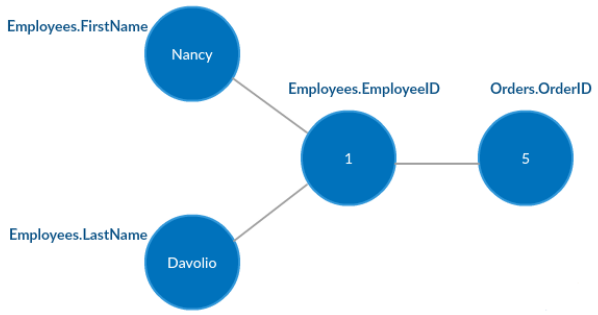

Figura B.1: Dependências de Chaves Estrangeiras

A existência de uma relação entre dois nós simboliza o fato de um utilizador, através das queries efetuadas até ao momento, conseguiu relacionar os respetivos valores dos atributos. Como num grafo uma aresta coneta apenas dois nós, e como um nó é utilizado quantas vezes o mesmo valor de um dado atributo for retornado pelas queries executadas, em cada relação é necessário manter um contexto a restringir a validade da mesma. Este contexto está então

associado a 2 restrições: a restrição do tuplo, designada de `tuple_restriction` e a restrição da cláusula where da query que permitiu esta associação, designada de `where_restriction`. Num contexto fictício, se a query q_2, apresentada na listagem 3.4, devolver os tuplos (EmployeeID, Salary) = {(1, 2000); (2, 2000)}, a respetiva representação gráfica é a apresentada na figura B.2.

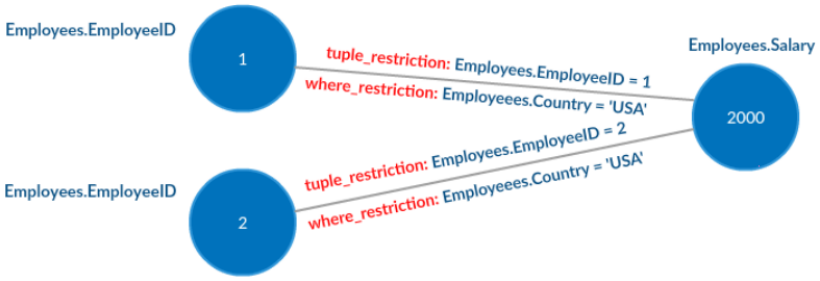

Figura B.2: Restrições de Tuplo e cláusula Where

Na figura B.3 pode-se perceber a importância da necessidade de se manter este contexto. Se no âmbito da execução da query

```
1 q1: select LastName, FirstName, Country from Employees;
```

forem retornados os tuplos (LastName, FirstName, Country) = { ('Davolio', 'Nancy', 'USA'), ('Dodsworth', 'Nancy', 'USA')}, a representação no grafo corresponderá à apresentada na figura supramencionada.

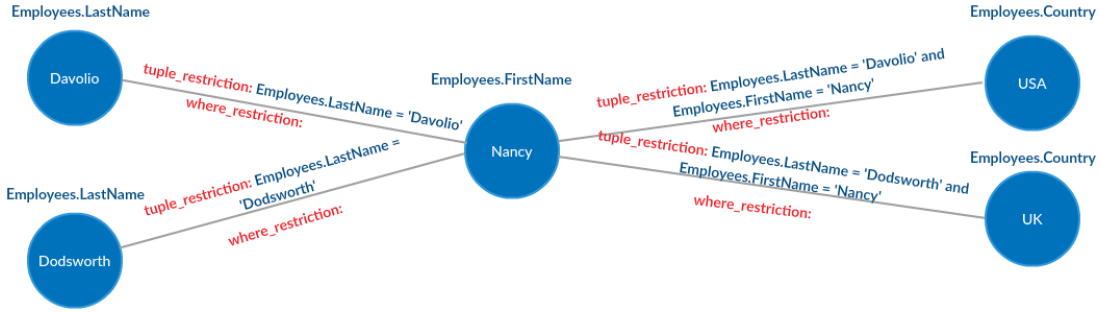

Figura B.3: Restrições de Tuplo e cláusula Where - motivação

Como se pode constatar, ao percorrer o grafo, para obter o valor do atributo Country associado ao empregado cujos (LastName, FirstName) = ('Davolio','Nancy') é necessário o contexto fornecido pela restrição de tuplo para obter o valor correto.

Posto isto, definindo a forma, como para as diferentes situações de possíveis queries que um utilizador possa submeter, como o valor de cada atributo num tuplo se coneta aos restantes.

Ao armazenar os tuplos de uma dada query, verifica-se se a restrição da cláusula where contém alguma condição que corresponda a uma igualdade. Caso exista, o nó correspondente ao valor do atributo é adicionado ao grafo juntamente com o tuplo. Posto isto, ao analisar a supramencionada query q_2, apresentada na listagem 3.4, e a respetiva representação, apresentadas na figura B.2, pode-se concluir que a mesma não está correta. A representação correta

dos resultados desta query é apresentada na figura B.4.

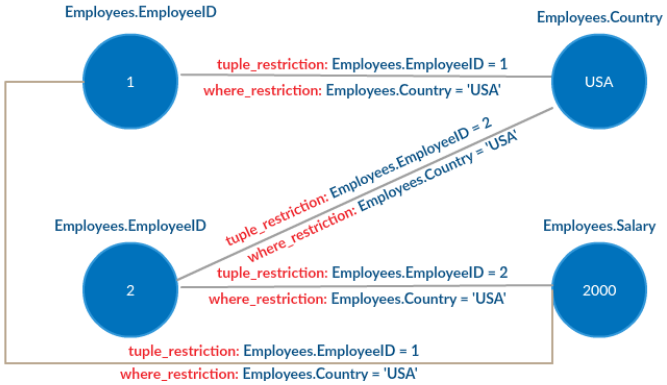

Figura B.4: Processamento de cláusula Where

De fato, apenas se pode fazer este processamento quando para qualquer combinação de condições válidas, a condição de igualdade esteja presente nessa mesma combinação. Recorrendo a dois exemplos para simplificar, quando a cláusula where é composta por c_1**:(A=1 or B<3)** as possíveis combinações que permitem validar a expressão correspondem a {A=1}, {B<3} ou {A=1, B<3}. Neste caso, a segunda combinação não possui a condição de igualdade, pelo que o processamento supramencionado não pode ser efetuado. Por outro lado, para a cláusula c_2**: (A=1 and (B>10 or B<3))** é possível efetuá-lo. Para todas as combinações possíveis que validem esta expressão {A=1, B>10} e {A=1, B<3} a condição de igualdade `A=1` é necessária.

Além do processamento supramencionado da cláusula where, foram também definidas algumas regras ao nível da conetividade dos nós associados aos atributos inquiridos por uma query. Elas têm como objetivo evitar que se criem, quando inquiridos N atributos, $\sum_{i=1}^{N-1} i$ relações entre os mesmos. Independentemente das relações criadas, a `where_restriction` toma sempre o valor da cláusula where da query.

Começando pelo caso mais simples, uma query pode inquirir apenas atributos de uma tabela. Neste caso a chave primária da mesma pode ser, ou não, inquirida juntamente com outros atributos. Quando a chave primária é inquirida, os valores da mesma servem de *ponto de ancoragem*. Quer isto dizer que, o(s) atributo(s) que a constituem são primeiramente conetados entre eles [1] e posteriormente, a partir do nó correspondente ao *último atributo* da chave, é efetuada a conexão aos restantes atributos comuns[2] inquiridos. As relações entre o nó que corresponde à chave primária e os nós que correspondem aos atributos comuns possui na `where_restriction` a restrição da query, e na `tuple_restriction` apenas a restrição correspondente à chave primária. É possível aplicar este tipo de conetividade uma vez que um tuplo é identificado inequivocamente pela chave da tabela. Como pode ser verificado, as representações apresentadas nas figuras B.4 e B.2 obedecem a este modelo.

Quando a chave da tabela não é inquirida, os atributos comuns são conetados também de forma ordenada. Neste caso, `tuple_restriction` vai *aumentando* à medida que os nós vão sendo conetados. Posto isto, para cada tuplo retornado pela query

[1] obedecendo à ordem pela qual foram declarados na criação da tabela
[2] designação atribuída aos atributos que não pertencem à chave da tabela

```
select LastName, FirstName, Salary from Employees;
```

a respetiva representação corresponde à apresentada na figura B.5.

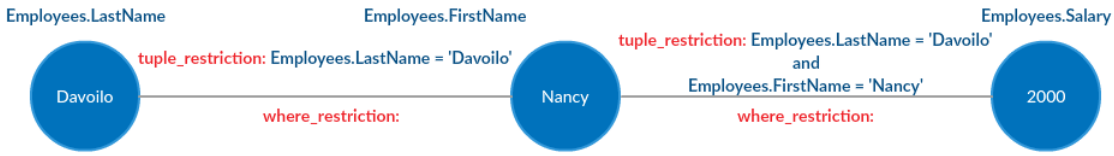

Figura B.5: Conetividade entre Atributos Comuns

Por fim, quando uma query inquire múltiplas tabelas, concetualmente o utilizador consegue relacionar os atributos inquiridos dessas mesmas tabelas. Neste caso, primeiramente são conetados os atributos de cada uma das tabelas entre si seguindo a estratégia já descrita. Posteriormente é representada a capacidade que o utilizador adquiriu de conseguir associar os valores dos atributos das múltiplas tabelas. Neste passo, é utilizada a `tuple_restriction` associada a cada uma das tabelas, conetando os *últimos* nós associados a cada tabela entre si. Nesta relação criada, `tuple_restriction` toma o valor da junção de `tuple_restriction` de cada uma das tabelas. De salientar que apenas é necessário efetuar esta associação quando não foi inquirida a chave estrangeira que consegue efetuar a associação entre tuplos de ambas as tabelas. Tendo esta estratégia em mente, a representação apresentada na figura B.6 está associada a um tuplo retornado pela query

```
select OrderID, OrderDate, Customers.ContactName from Orders join Customers on
    Orders.CustomerID = Customer.CustomerID;
```

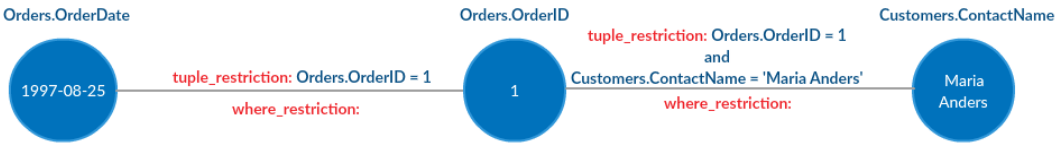

Figura B.6: Conetividade entre Atributos de Tabelas distintas

A solução baseada em grafos, foi uma primeira abordagem natural devido ao contato prévio com este tipo de paradigma. No entanto, tal como foi possível constatar ao longo desta seção, o fato de neste tipo de solução as arestas apenas poderem conetar dois nós apresenta limitações quando o tuplo é composto por mais do que dois atributos. Para as solucionar necessário desenvolver as estratégias descritas:

- a necessidade de manter o contexto nas relações recorrendo às restrições de tuplo e de cláusula where;
- a criação de algumas regras para a representar a conetividade entre os atributos inquiridos no tuplo, para limitar o número de relações criadas.

Além destas limitações e perante a solução concetual apresentada neste anexo, após uma análise mais cuidada à mesma pode-se constatar que ela possui ainda uma grande limitação. Devido ao fato de o contexto em que cada tuplo é obtido ser armazenado nas relações entre

os nós do mesmo, quando uma query apenas inquire um atributo (não sendo criada nenhuma relação) não é possível saber que os valores retornados estão associados ao contexto da cláusula where da query em questão. Na seção 3.3 é abordada a importância deste aspeto.

Para resolver este problema poder-se-ia alterar a forma como o contexto era armazenado: mantendo o restrição de tuplo armazenada nas relações, podia-se passar a restrição da cláusula where em que o nó foi obtido para o nó em si. Assim, quando ocorresse o caso de uma query apenas inquirir um atributo podia-se, ao verificar para cada nó associado a esse mesmo atributo, averiguar se o mesmo estava associado ao contexto da query. Concetualmente, esta alteração pode ser representada na figura B.7

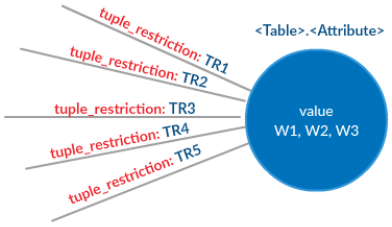

Figura B.7: Alteração - armazenamento do contexto

Uma segunda solução deste problema poderia passar por criar uma ligação entre os nós pertencentes ao tuplo retornado a um nó especial (nó de query). Para evitar que se criassem ligações a todos os nós que compunham o tuplo, poder-se-ia recorrer às mesmas estratégias utilizadas previamente; para cada tabela inquirida conetava-se o nó de query ao último atributo da chave primária (ou candidata) da mesma, caso tivesse sido inquirida, ou ao último atributo comum, caso contrário. Estas ligações dispensariam a restrição da cláusula where e a restrição de tuplo corresponderia à restrição completa do tuplo pertencente a cada tabela.

A representação desta segunda alternativa é mostrada na figura B.8.

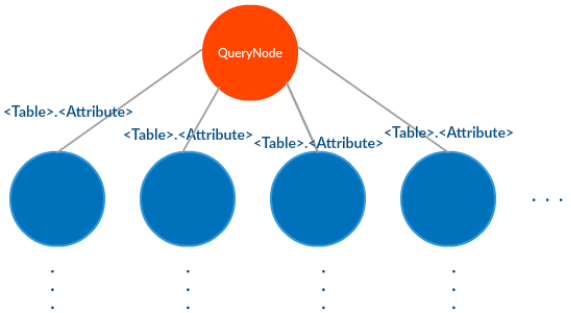

Figura B.8: Alteração - adição de nó de query

Perante esta segunda abordagem surgiu solução do armazenamento da informação recorrendo a hipergrafos.

Execução de queries - solução baseada em grafos

No âmbito da solução baseada em grafos, utilizou-se a ferramenta `Neo4j` para armazenar a informação inquirida por cada query. A escolha recaiu sobre esta base de dados devido à simplicidade, desempenho e robustez da mesma.

Cada query era executada sobre a base de dados original, sendo para tal necessário recorrer a técnicas de reescrita de queries, para que as alterações à estrutura da mesma apresentadas no âmbito da subseção 3.1.2 fossem refletidas na informação efetivamente obtida. Estas alterações consistiam apenas em adicionar os atributos necessários à cláusula *select* da mesma.

Relativamente à forma como a informação é armazenada, é necessário ter em conta que a ferramenta Neo4J apenas permite o armazenamento de tipos de dados primitivos (ou listas desses tipos de dados), tanto nos nós como nas relações. Esta informação é adicionada a cada nó/relação sob a forma de propriedade dos mesmos. Adicionalmente, permitem que sejam definidas etiquetas (*labels*), que podem ser utilizadas nas duas entidades supramencionadas. De fato, a cada nó/relação podem estar associadas múltiplas etiquetas.

Posto isto, ao nível dos nós, foi definida a propriedade *value* onde era armazenado cada valor do respetivo atributo. Em termos da identificação dos nós, as etiquetas foram definidas sob a forma do nome normalizado do atributo <table_name>.<attribute_name>, apresentada na seção 3.2.

Adicionalmente, ao nível das relações, tanto as restrições de tuplo como de cláusula where (apresentadas na seção `Solução de armazenamento de informação recorrendo a grafos` do Anexo `B`), que eram armazenadas na estrutura `Restriction` (definida na subseção `Agregação de Expressões Simples` do Anexo `C`) necessitavam de uma representação sob a forma de String para poderem ser armazenadas no grafo. Posteriormente sempre que se pretendesse percorrer o grafo, para fazer a interpretação destas duas cláusulas, era necessário recontruí-las. Este processo, devido à quantidade de informação com que se pode estar a lidar e, consequentemente, o número de relações que necessárias, introduzia um *overhead* significativo ao tempo de execução desta solução.

Desenvolvimento de coleções suportadas por ficheiros

Perante a representação do armazenamento da informação em coleções apresentada na figura supramencionada 4.5, foi necessário primeiramente definir uma forma de identificar cada coleção. Para tal, de forma a obter um identificador único para cada coleção, recorreu-se a uma medida de tempo em nanosegundos recorrendo ao método `System.nanoTime()`[3].

Posto isto, cada coleção foi definida utilizando a nomenclatura

<collection_identifier>_<result_type>

em que <collection_identifier> corresponde ao identificador único da coleção e <result_ type> corresponde ao tipo de resultados obtidos que a coleção armazena (recorrendo à notação apresentada na figura 4.5). Esta identificação era utilizada ao nível do nome do ficheiro da respetiva coleção.

Relativamente à informação efetivamente armazenada, eram depositados os valores dos atributos obtidos bem como as queries através das quais tinham sido obtidos. Adicionalmente, era necessário, aquando da existência de uma referência para uma coleção filha, armazenar o identificador dessa mesma coleção. Esta informação era armazenada recorrendo à estrutura `SingleExpression`.

Cada query era então acondicionada na estrutura SingleExpression definindo:

[3]https://docs.oracle.com/javase/8/docs/api/java/lang/System.html#nanoTime

- <attribute_name> com a macro `QUERY_ID`;
- <relation> `EQUALS`;
- <attribute_value> com a declaração da query submetida pelo utilizador.

Por seu lado, o identificador da coleção filha era acondicionado na estrutura SingleExpression definindo:

- <attribute_name> com a macro `COLLECTION_NAME`;
- <relation> `EQUALS`;
- <attribute_value> com o valor o nome do ficheiro da respetiva coleção.

Por fim, convém referir que no âmbito do armazenamento destas três entidades, por questões de organização, eram obedecidos os seguintes princípios:

1. a informação era armazenada de forma ordenada, onde, por cada query atravessada no âmbito do caminho era primeiramente armazenado o identificador da query e apenas depois os novos valores dos atributos obtidos através dessa mesma query;
2. em caso de existência, a referência para a coleção filha correspondia ao último componente armazenado no contexto de uma linha.

Posto isto, a representação da informação armazenada sob a forma de coleções suportadas por ficheiros, é apresentada na figura B.9.

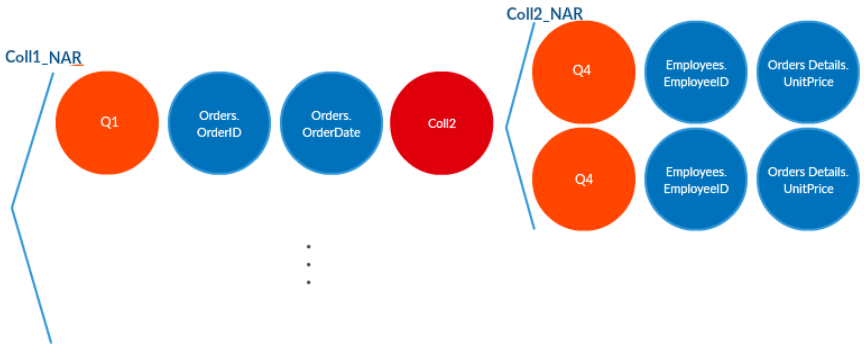

Figura B.9: Representação de armazenamento de informação recorrendo a coleções suportadas por ficheiros

Em termos de dados efetivamente armazenados em cada coleção (ficheiro), eram apenas escritas as instâncias de cada SingleExpression para o ficheiro. No entanto, para conseguir distinguir as várias linhas da coleção (onde cada linha correspondia a um ramo distinto do contexto corrente), foi necessário utilizar delimitadores que permitissem à solução desenvolvida no âmbito desta dissertação perceber claramente onde *começavam* e *acabavam* cada uma destas linhas. A primeira abordagem a este problema passou pela definição de dois delimitadores especiais que indicavam o início e o fim de cada linha. Na segunda abordagem, antes de cada linha começar, era escrito no ficheiro o número de instâncias de SingleExpressions

que compunham a *próxima* linha da coleção. Assim, enquanto um ficheiro não acabasse, após ler o número de objetos (SingleExpresion) indicados previamente, sabia-se que de seguida apareceria outro número a indicar o número de SingleExpressions que compunham a próxima linha da coleção.

Anexo C

Unidades básicas de Informação

De acordo com o objetivo desta dissertação, onde se tem que comparar valores de atributos e verificar a validade dos mesmos no âmbito das condições presentes nas políticas de acesso (definidas sob a forma de queries SQL, tal como mencionado em 3.1.1), sentiu-se a necessidade de primeiramente criar uma estrutura que suportasse cada expressão condicional. Posteriormente, tornou-se necessário também desenvolver uma estrutura que suportasse a agregação dessas mesmas expressões. Estas duas estruturas serão descritas nas subseções `Expressão Simples` e `Agregação de Expressões Simples` do Anexo `C`, respetivamente.

Expressão Simples

A estrutura desenvolvida para suportar cada expressão condicional foi designada de `SingleExpression`. Recorrendo a ela pode-se armazenar (1) o atributo a que a expressão se refere, (2) a relação condicional e (3) o valor que serve de termo de comparação. Este armazenamento pode ser representado por

<attribute_name> <relation> <attribute_value>

Neste contexto, o nome do atributo é definido da forma apresentada na seção 3.2, em que os nomes da tabela e do atributo devem corresponder aos originais. No âmbito desta estrutura são suportadas as relações $\{=, !=, >, <, >=, <=$, between e in$\}$. Por fim, os valores dos atributos podem ser dos tipos {boolean, byte, short, int, long, float, double, date, char, String e byte[]}.

Além do armazenamento, é também ao nível da estrutura `SingleExpression` que se realizam operações importantes no contexto da avaliação como a interseção e validação de expressões. Estas operações aplicam-se sobre expressões relativas ao mesmo atributo. A operação de interseção é simples e intuitiva. Relativamente à operação de validação, seja R_x o conjunto dos possíveis valores assocciados a uma expressão X e R_y o conjunto dos valores associados à expressão Y; X é válida no contexto de Y se $\forall_v \in R_x \Rightarrow v \in R_y \equiv X \subset Y$. Posto isto, é fácil perceber que a expressão condicional `Employees.BirthDate > '01-01-1990'` é válida no âmbito da expressão `Employees.BirthDate > '01-01-1985'`.

A título de exemplo são apresentadas de seguida algumas expressões condicionais:

- Employees.Country = 'USA';
- Employees.FirstName = 'Nancy';
- Orders.OrderDate > '01-02-1997';

- Employees.EmployeeID between 5 and 10.

Agregação de Expressões Simples

No âmbito desta solução, a agregação de múltiplas expressões condicionais, é efetuada ao nível da estrutura `Restriction`. Esta estrutura armazena estas expressões sob a forma de uma árvore binária em que as folhas da mesma correspondem a expressões condicionais enquanto os restantes nós da árvore aos operadores lógicos existentes `and` e `or`.

De fato, estes operadores lógicos são também armazenados ao nível da estrutura SingleExpression; correspondendo esta vertente à segunda em que a respetiva estrutura pode ser utilizada.

Posto isto, para a cláusula where presente na query apresentada na listagem C.1, a representação da mesma corresponde à apresentada na figura C.1.

```
1 select EmployeeID, LastName, FirstName from Employees where Country = 'USA' and
      BirthDate > '01-01-1985';
```

Listagem C.1: Definição de query

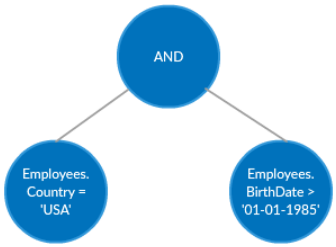

Figura C.1: Representação da estrutura Restriction

Esta estrutura permite também posteriormente efetuar operações de avaliação tendo como pressupostos outra `Restriction` ou um conjunto de `SingleExpressions`. Uma restrição R_1 é válida no contexto de outra R_2 caso para todas as possíveis combinações válidas de *SingleExpressions* em R_1 sejam também válidas em R_2. Ou seja, $R_1 \subset R_2$.

Ao nível desta estrutura é- também possível obter a Restriction complementar, cuja importância será evidenciada no âmbito da subseção 4.7.5. De acordo com a teoria de conjuntos, tal como descrito em [61], o complemento de uma uniãp corresponde à interseção dos complementos enquanto que o complemento de uma interseção corresponde à união dos complementos:

- $(A \cup B)^C = A^C \cap B^C$
- $(A \cap B)^C = A^C \cup B^C$

Posto isto, tendo em conta a restrição dada pela expressão

$$(\text{Employees.Salary} > 2000 \cap \text{Employees.FirstName} = \text{'Nancy'})$$

a sua restrição complementar corresponde a

$$(\text{Employees.Salary} <= 2000 \cup \text{Employees.FirstName} != \text{'Nancy'})$$

Pode-se então perceber, que ao nível de `SingleExpression` precisou-se também de implementar uma operação que permita obter o complemento de uma expressão condicional.

Anexo D

Processamento de uma query recorrendo a FoundationDB

Esta ferramenta interpreta uma query e devolve a estrutura da mesma sob a forma de uma estrutura em árvore, sob o padrão de design *visitor*[4]. Cada nó desta estrutura corresponde a um elemento da query; atributos e tabelas inquiridos, condições presentes nas restrições das cláusulas where e outros elementos descritivos do tipo dos subsequentemente apresentados. O nós são devolvidos por esta ferramenta sob a forma `pré-ordem`[5], onde (1) é retornada a informação do nó raíz, (2) o subramo da esquerda e, por fim, (3) o subramo da direita . Na figura D.1, é apresentado um exemplo da forma como esta ferramenta processa a query presente na listagem D.1.

```
1 select FirstName FROM Employees where EmployeeID = 3;
```

Listagem D.1: Definição de query processada por FoundationDB

Como se pode perceber, esta ferramenta não retorna uma estrutura `Query` completa, mas sim os vários componentes da mesma. Cabe então às aplicações terceiras (que recorrem a esta ferramenta) criar as suas próprias estruturas e preenchê-las à medida que vão obtendo cada nó retornado pela ferramenta `FoundationDB`. Para possibilitar este processamento por estas aplicações cada nó devolvido pela ferramenta `FoundationDB` implementa a *interface* `Visitable` [6] enquanto que as soluções que recorrem a esta ferramenta necessitam de implementar a *interface* `Visitor`[7] para poderem aceder a cada nó da query segundo a ordem supramencionada. Na listagem D.2 é apresentado um excerto de código que permite o devido processamento de cada nó da query. Pode-se verificar que é necessário tratar individualmente cada instância do nó de forma a preencher/completar a estrutura de cada *query*.

```
1  @Override
2  public Visitable visit(Visitable node) throws StandardException {
3      if (node instanceof FromList) {
4          //node processing goes here
5      } else if (node instanceof BinaryLogicalOperatorNode) { //OR/AND
6          //node processing goes here
7      } else if (node instanceof BinaryArithmeticOperatorNode) { //= / < / > /
       ...
8          //node processing goes here
9      } else if (node instanceof FromBaseTable) { //FROM...
10         //node processing goes here
```

[4]https://en.wikipedia.org/wiki/Visitor_pattern
[5]https://en.wikipedia.org/wiki/Tree_traversal
[6]http://javadox.com/com.foundationdb/fdb-sql-parser/1.1.0/com/foundationdb/sql/parser/Visitable.html
[7]http://javadox.com/com.foundationdb/fdb-sql-parser/1.1.0/com/foundationdb/sql/parser/Visitor.html

```
11      } else if (node instanceof ColumnReference) { // attribute names (select
        and where clauses)
12          //node processing goes here
13      } else if  ...
14  }
```

Listagem D.2: Processamento de uma Query recorrendo a FoundationDB por parte de uma aplicação *third-party*

```
com.foundationdb.sql.parser.CursorNode@378fd1ac
name: null
updateMode: UNSPECIFIED
statementType: SELECT
resultSet:
        com.foundationdb.sql.parser.SelectNode@49097b5d
        isDistinct: false
        resultColumns:
                com.foundationdb.sql.parser.ResultColumnList@6e2c634b

                [0]:
                com.foundationdb.sql.parser.ResultColumn@37a71e93
                exposedName: firstname
                name: firstname
                tableName: null
                isDefaultColumn: false
                type: null
                expression:
                        com.foundationdb.sql.parser.ColumnReference@7e6cbb7a
                        columnName: firstname
                        tableName: null
                        type: null
        fromList:
                com.foundationdb.sql.parser.FromList@7c3df479

                [0]:
                com.foundationdb.sql.parser.FromBaseTable@7106e68e
                tableName: employees
                updateOrDelete: null
                null
                correlation Name: null
                null
        whereClause:
                com.foundationdb.sql.parser.BinaryRelationalOperatorNode@7eda2dbb
                operator: =
                methodName: equals
                type: null
                leftOperand:
                        com.foundationdb.sql.parser.ColumnReference@6576fe71
                        columnName: employeeid
                        tableName: null
                        type: null
                rightOperand:
                        com.foundationdb.sql.parser.NumericConstantNode@76fb509a
                        value: 3
                        type: INTEGER NOT NULL
```

Figura D.1: Resultado do processamento de uma Query recorrendo a FoundationDB

Como se pode depreender da listagem D.2, as classes `FromList`, `BinaryLogicalOperatorNode`, `BinaryArithmeticOperatorNode`, `FromBaseTable`, `ColumnReference` correspondem a classes que implementam a interface Visitable e que estão associadas a elementos distintos de uma query e que requerem, cada uma, um processamento adequado. A título de exemplo, como se pôde observar na imagem D.1, o tipo de dados `ColumnReference`, pode ser retornado no âmbito de uma cláusula select bem como no âmbito de uma cláusula where. Quer por esta razão, quer pela ocorrência de subqueries, torna-se então necessário, a cada nó recebido saber a fase em que se encontra do processamento da query.

Anexo E

Algoritmo de normalização de nomes de atributos

O algoritmo da operação da normalização do nome do atributo é apresentado no algoritmo 6.

Data: originalAttributeName
Result: normalizedAtributeName

```
1  currentTable ← originalAttributeName;
2  currentAttribute ← originalAttributeName;
3  changed ← true;
4  while changed do
5  |   changed ← false;
6  |   if hasForeignKey(currentTable, currentAttribtue) then
7  |   |   currentTable ← currentTableGetForeignKeyTable(currentAttribute);
8  |   |   currentAttribute ← currentTableGetForeignKeyAttribute(currentAttribute);
9  |   |   changed ← true;
10 |   end
11 end
12 normalizedAtributeName ← currenteTabe + "." + currentAttribute
```

Algoritmo 6: Normalização de nomes de atributos

Como se pode constatar, esta solução acautela a utilização do mesmo atributo numa cadeia de chaves estrangeira, na medida em que, através do ciclo *while*, enquanto os nomes do atributo e da tabela obtidos na iteração anterior corresponderem a uma chave estrangeira, continua-se a obter a respetiva referência.

Algoritmo de execução de queries submetidas por um utilizador

Como se pode perceber através da análise do algoritmo 7, para obter o tuplo inquirido por uma query (submetida pelo utilizador) é necessário obter o tuplo completo das tabelas inquiridas. Entende-se como `tuplo completo das tabelas` o conjunto de atributos das tabelas associados por uma operação de *inner join*, ou seja, onde existe correspondência entre os valores dos atributos das respetivas tabelas. Após a obtenção do tuplo completo, verifica-se a sua validade no âmbito da cláusula where. Caso o mesmo seja válido, o tuplo é adicionado ao conjunto de tuplos retornados, removendo os atributos que não estão presentes na cláusula select da query.

```
1 for tuple accross the tables inquired do
2 |   if tuple is valid in where clause restriction context then
3 |   |   inquiredTupleValues ← getInquiredAttributesValues(tuple);
4 |   |   tulesReturned ← inquiredTupleValues;
5 |   end
6 end
```

Algoritmo 7: Execução de Query submetida por Utilizador

Algoritmo do método `userAccessedThisTuple`

```
   Data: currentCompleteTuple
   Result: queriedAllValues
1  attributesToSearchFor ← qill.getAttributes();
2  for query in executedQueries do
3  |   if query.getAttributes() contains any attributesToSearchFor then
4  |   |   add attributes inquired by qill to attributesToUseOnAdminCount;
5  |   |   add atributes inquired by query to attributesToUseOnAdminCount;
6  |   |   add query's where clause to restrictions;
7  |   |   if getAdminCount(attributesToUseOnAdminCount, restrictions,
   |   |    currentCompleteTuple > 0) then
8  |   |   |   remove query's inquired attributes from attributesToSearchFor
9  |   |   end
10 |   end
11 end
12 queriedAllValues ← attributesToSearchFor.isEmpty()
```

Algoritmo 8: Modo de funcionamento do método `userAccessedThisTuple`

Através da análise do algoritmo 8, pode-se perceber que para cada query submetida pelo utilizador e que contenha atributos que ainda não foram *encontrados* nas iterações anteriores, se tenta verificar se os mesmos foram obtidos no âmbito da presente iteração. Se no final todos os valores de atributos foram obtidos pelo conjunto de queries submetidas pelo utilizador, considera-se o respetivo tuplo sensível.

Algoritmo de Determinação de Caminhos

O algoritmo 9 apresenta o processo base da determinação de caminhos.

Como se pode constatar, este corresponde a um algoritmo recursivo. Os caminhos começam a ser formados a partir do fim, sendo atribuído ao último passo o nome do atributo crítico que se pretende encontrar. A partir daí, a cada chamada recursiva, são procurados nos atributos inquiridos pelas queries que ainda não foram utilizadas, um subconjunto de atributos que contenha o(s) atributo(s) que pertence(m) ao primeiro passo do caminho apresentado. Após acrescentar ao caminho um novo primeiro passo e remover a query que foi utilizada do conjunto das queries ainda não utilizadas, é efetuada uma nova chamada recursiva.

```
1  getPathsToAttributesSearchingFor(initialPath, queriesToAnalyze){
2  begin
3      add all queries on queriesToAnalyze to set newQueries;
4      add initialPath to set resultingPaths;
5      foreach query in queriesToAnalyze do
6          if initialath's first hop contains any attribute from query then
7              remove query from set newQueries;
8              newPath ← initialPath;
9              add query as first hop to set newPath;
10             add newPath to newPaths;
11         end
12     end
13     if newPaths isn't empty then
14         foreach newPath in newPaths do
15             add result from getPathsToAttributesSearchingFor(newPath,
                newQueries) to resultingPaths;
16         end
17     end
18     return resultingPaths;
19 end
20 }
```

Algoritmo 9: Obtenção de Caminhos

Algoritmo de determinação de tipos de resultados de um atributo inquirido

Pode-se perceber que o tipo de resultado de um atributo at1, em que a tabela t1 corresponde à sua tabela *original*, será ALL_RELATED se a projeção dos atributos inquiridos na query com os atributos que compõem o contexto corrente contiver a chave de uma tabela t2 cuja relação com t1 seja de 1:1[8] ou N:1[9]. Alternativamente, se a relação supramencionada for de 1:N[10], apenas se pode assumir que o resultado será ALL_RELATED caso t1 não apareça em nenhum dos caminhos entre t2 e as restantes tabelas que contêm os atributos do contexto corrente. Por fim, é também fácil de perceber que se o atributo inquirido já fizer parte do contexto atual, todos os valores obtidos serão ALL_RELATED.

Caso as situações acima descritas não se verifiquem, o tipo de resultado de at1 será NOT_ALL_RELATED.

Algoritmo de verificação da necessidade de remoção de linhas sobrepostas de coleções

O processamento é apresentado no algoritmo 11.

[8] p.ex. quando são as mesmas tabelas
[9] p.ex. t1=Orders; t2=Customers
[10] p.ex. t1=Customers; t2=Orders

```
Data: name from attributeToGetResultType, names from
      commonAttributesInQueryAndContext
Result: attribute values result type
1  if commonAttributesInQueryAndContext contains attributeToGetResultType then
2  |  return ALL_RELATED;
3  end
4  keyTables ← tables that contain unique keys formed by attributes on
     commonAttributesInQueryAndContext;
5  tablesOnContext ← tables that contain attributes on
     commonAttributesInQueryAndContext;
6  attributeTable ← original table from attribute attributeToGetResultType;
7  foreach table in keyTables do
8  |  if relation between table and attributeTable equals 1:1 or N:1 then
9  |  |  return ALL_RELATED;
10 |  end
11 |  else
12 |  |  if paths between table table and the tables on set tablesOnContext don't
   |  |    contain attributeTable then
13 |  |  |  return ALL_RELATED;
14 |  |  end
15 |  end
16 end
17 return NOT_ALL_RELATED;
```

Algoritmo 10: Determinação do tipo de resultados de um atributo inquirido

```
1  needToRemoveLink(context, line){
2  begin
3  |  store in val1 number of valid tuples in context;
4  |  counter ← 0;
5  |  foreach Ctx_aggr<i> in Ctx_un do
6  |  |  counter += number of valid tuples in context and Ctx_aggr<i> and invalid in
   |  |    Ctx_aggr<j>, for j < i;
7  |  end
8  |  return counter == val1;
9  end
10 }
```

Algoritmo 11: Verificação da necessidade de remoção de linhas sobrepostas de coleções

---

Tal como noutros casos, recorre-se ao método `getAdminCount` para determinar se toda a informação obtida em Ctx_i é obtida em Ctx_{un}: primeiramente obtém-se o número de tuplos válidos no âmbito de Ctx_i; de seguida, para cada $\text{Ctx}_{aggr<i>} \in \text{Ctx}_{un}$, obtém-se o número de tuplos válidos em Ctx_i e $\text{Ctx}_{aggr<i>}$ e inválidos no âmbito de $\text{Ctx}_{aggr<j>}$ tal que j < i. Posteriormente, compara-se o valor obtido no âmbito de Ctx_i com a soma dos valores obtidos no âmbito de cada $\text{Ctx}_{aggr<i>}$; se o resultado corresponder a uma igualdade sabe-se que Ctx_i

é abrangido por Ctx_{un} e, consequentemente, pode-se remover a respetiva linha.

Para o exemplo expandido apresentado anteriormente no âmbito da listagem 4.19, no âmbito da linha da coleção `Collection1_REL`, apresentada na figura 4.11, sabe-se que a mesma é utilizada no âmbito de Ctx_{aggr1}, Ctx_{aggr2} e Ctx_{aggr3}.

Posto isto, suponha-se que (CustomerID = 'cid1') está associado apenas a Orders cujo (OrderDate = 'date1') identificadas por (OrderID = 1) e (OrderID = 2). Para os valores { (CustomerID = 'cid1'), (CompanyName = 'company1'), (ContactName = 'contact1'), (OrderDate = 'date1')} começa-se por obter o número de tuplos válidos para o contexto da respetiva linha que, neste caso não é restringido por nenhuma cláusula where; cujo resultado corresponde a dois.

Posto isto, no âmbito de Ctx_{aggr1} obtém-se o número de tuplos válidos para os valores supramencionados de CustomerID, CompanyName, ContactName e OrderDate e para (EmployeeID < 4). O contador obtido corresponde a um.

No âmbito de Ctx_{aggr2}, obtém-se o número de tuplos válidos para ((EmployeeID >= 4) and (EmployeeID between 3 and 6)), cujo resultado é zero.

Por fim, para Ctx_{aggr3}, obtém-se o número de tuplos válidos para ((EmployeeID >= 4) and ((EmployeeID < 3) or (EmployeeID > 6))) and (EmployeeID >= 5), cujo resultado é 1.

Como (2 = 1 + 1), é detetado que pode remover a respetiva linha.

Algoritmo de remoção de resultados sobrepostos

O algoritmo 12 apresenta a forma como conjuntos de resultados sobrepostos são removidos.

```
1  removeOverlappedCollection(collection, context){
2  begin
3      foreach line in collection do
4          newContext ← context;
5          needToRemove ← needToRemoveLink(newContext, line);
6          add information from line to newContext;
7          if collection has a child collection then
8              removeOverlappedCollection(childCollection, newContext);
9          end
10         else
11             if needToRemove is true then
12                 remove line from collection;
13             end
14         end
15     end
16     if collection doesn't have any valid tuple in context's context then
17         collection.delete();
18     end
19 end
20 }
```

Algoritmo 12: Remoção de coleções de resultados sobrepostos

De fato, pode-se perceber que uma coleção não é eliminada apenas caso tenham sido apagadas todas as linhas da mesma. Por questões de otimização, uma porção do processamento efetuado na normalização dos resultados obtidos, descrito em 4.7.8, é realizada a este nível. Após a remoção das linhas reutilizadas é verificado se sobram algumas que sejam válidas no contexto corrente e caso nenhuma o seja, a coleção é removida. O motivo de este processamento ser realizado nesta fase prende-se com o fato de à medida que se percorre cada linha da coleção, adicionalmente, ser necessário verificar se a mesma corresponde a informação válida no respetivo contexto ou não. Posteriormente, se não se remover pelo menos uma linha que contém informação considerada válida sabe-se que a respetiva coleção, após este processamento, contém pelo menos uma linha válida. Recorrendo a esta verificação evita-se que no processo de normalização se tenha que analisar a coleção de novamente quando a mesma contém apenas linhas inválidas.

Anexo F

```
TUPLE: (`customers`.`customerid` = 'ANATR')    (`customers`.`companyname` = 'Ana Trujillo Emparedados y helados')    (`customers`.`contactname` = 'An
Searching for [`order details`.`unitprice`]
Already have values from attributes [`customers`.`customerid`, `customers`.`contactname`, `customers`.`companyname`]
Executing path:
0. select ContactTitle, CustomerID, City from Customers
1. select City, OrderID, OrderDate, RequiredDate, EmployeeID from `Orders` join Customers on `Orders`.`CustomerID` = `Customers`.`CustomerID`
2. select UnitPrice, RequiredDate from `Order details` join Orders on `Order details`.`OrderID` = `Orders`.`OrderID` where EmployeeID <= 6

        Searching for [`orders`.`orderid`, `orders`.`orderdate`, `orders`.`requireddate`, `employees`.`employeeid`]
        Already have values from attributes [`customers`.`customerid`, `customers`.`contacttitle`, `customers`.`contactname`, `customers`.`companyname`,
        Executing path:
        0. select City, OrderID, OrderDate, RequiredDate, EmployeeID from `Orders` join Customers on `Orders`.`CustomerID` = `Customers`.`CustomerID`

        Filtering obtained results through the usage of where clauses
        Exiting results filtering through the usage of where clauses
        Executing path:
        0. select ContactTitle, OrderID from `Orders` join Customers on `Orders`.`CustomerID` = `Customers`.`CustomerID`

        Filtering obtained results through the usage of where clauses
        Exiting results filtering through the usage of where clauses
        Aggregating results from current path with results from previous ones
        Removing overlapped collections
        Removing known invalid tuples
        Normalizing collections
        Exiting Search for [`orders`.`orderid`, `orders`.`orderdate`, `orders`.`requireddate`, `employees`.`employeeid`]
    Filtering obtained results through the usage of where clauses
    Exiting results filtering through the usage of where clauses
    Removing overlapped collections
    Removing known invalid tuples
    Normalizing collections
    Exiting Search for [`order details`.`unitprice`]
```

Figura F.1: Excerto de relatório criado no processo de avaliação

```
1355522928846 57_REL {
    1 (queryID = '1')(queryID = '3')(`customers`.`customerid` = 'ALFKI')(`employees`.`employeeid` = '1')(`orders`.`requireddate` = '1998-02-12 00:00:
    1355522931644145_NAR_1355522928846 57 {
            1 (queryID = '2')(queryID = '4')(queryID = '5')(`orders`.`shippeddate` = '1998-01-21 00:00:00.0')(`employees`.`employeeid` = '1')(`order
            2 (queryID = '2')(queryID = '4')(queryID = '5')(`orders`.`shippeddate` = '1998-01-21 00:00:00.0')(`employees`.`employeeid` = '1')(`order
            3 (queryID = '2')(queryID = '4')(queryID = '5')(`orders`.`shippeddate` = '1998-01-19 00:00:00.0')(`order details`.`unitprice` = '123.7900
            4 (queryID = '2')(queryID = '4')(queryID = '5')(`orders`.`shippeddate` = '1998-01-19 00:00:00.0')(`employees`.`employeeid` = '1')(`orders
        }
    2 (queryID = '1')(queryID = '3')(`orders`.`orderid` = '10952')(`customers`.`customerid` = 'ALFKI')(`orders`.`requireddate` = '1998-04-27 00:00:00
    1355523795895152_NAR_1355522928846 57 {
            1 (queryID = '2')(queryID = '4')(queryID = '5')(`order details`.`unitprice` = '25.0000')(`employees`.`employeeid` = '1')(`orders`.`orderd
            2 (queryID = '2')(queryID = '4')(queryID = '5')(`employees`.`employeeid` = '1')(`orders`.`orderdate` = '1998-03-16 00:00:00.0')(`order de
            3 (queryID = '2')(queryID = '4')(queryID = '5')(`order details`.`unitprice` = '22.0000')(`employees`.`employeeid` = '1')(`orders`.`shippe
        }
    3 (queryID = '1')(queryID = '3')(queryID = '2')(queryID = '4')(queryID = '5')(`order details`.`unitprice` = '13.2500')(`customers`.`customerid` =
    4 (queryID = '1')(queryID = '3')(queryID = '2')(queryID = '4')(queryID = '5')(`customers`.`customerid` = 'ALFKI')(`employees`.`employeeid` = '3')
    5 (queryID = '1')(queryID = '3')(`orders`.`orderdate` = '1997-08-25 00:00:00.0')(`customers`.`customerid` = 'ALFKI')(`orders`.`requireddate` = '1
    1355511095219015_NAR_1355522928846 57 {
            1 (queryID = '2')(`orders`.`orderdate` = '1997-08-25 00:00:00.0')(`order details`.`unitprice` = '45.6000')
            2 (queryID = '2')(`orders`.`orderdate` = '1997-08-25 00:00:00.0')(`order details`.`unitprice` = '18.0000')
            3 (queryID = '2')(`orders`.`orderdate` = '1997-08-25 00:00:00.0')(`order details`.`unitprice` = '12.0000')
            4 (queryID = '2')(`orders`.`orderdate` = '1997-08-25 00:00:00.0')(`order details`.`unitprice` = '62.5000')
            5 (queryID = '2')(`orders`.`orderdate` = '1997-08-25 00:00:00.0')(`order details`.`unitprice` = '46.0000')
        }
    6 (queryID = '1')(queryID = '3')(`customers`.`customerid` = 'ALFKI')(`employees`.`employeeid` = '4')(`orders`.`orderdate` = '1997-10-03 00:00:00.
    1355111171406 78_NAR_1355522928846 57 {
            1 (queryID = '2')(`order details`.`unitprice` = '18.0000')(`orders`.`orderdate` = '1997-10-03 00:00:00.0')
            2 (queryID = '2')(`order details`.`unitprice` = '123.7900')(`orders`.`orderdate` = '1997-10-03 00:00:00.0')
            3 (queryID = '2')(`orders`.`orderdate` = '1997-10-03 00:00:00.0')(`order details`.`unitprice` = '46.0000')
            4 (queryID = '2')(`order details`.`unitprice` = '19.4500')(`orders`.`orderdate` = '1997-10-03 00:00:00.0')
            5 (queryID = '2')(`order details`.`unitprice` = '49.3000')(`orders`.`orderdate` = '1997-10-03 00:00:00.0')
            6 (queryID = '2')(`orders`.`orderdate` = '1997-10-03 00:00:00.0')(`order details`.`unitprice` = '43.9000')
        }
    7 (queryID = '1')(queryID = '3')(`customers`.`customerid` = 'ALFKI')(`orders`.`orderid` = '10702')(`orders`.`orderdate` = '1997-10-13 00:00:00.0'
    1355111600638 24_NAR_1355522928846 57 {
                .
                .
                .
```

Figura F.2: Exemplo de Coleção Processada

```
770659049036O4_REL {
    1 (queryID = '4')(queryID = '2')(`customers`.`city` = 'Berlin')(`orders`.`orderdate` = '1997-08-25 00:00:00.0')(`customers`
    770659073278l2_NAR_770659049036O4 {
        1 (queryID = '5')(`orders`.`requireddate` = '1997-09-22 00:00:00.0')(`order details`.`unitprice` = '45.6000')
        2 (queryID = '5')(`orders`.`requireddate` = '1997-09-22 00:00:00.0')(`order details`.`unitprice` = '18.0000')
        3 (queryID = '5')(`orders`.`requireddate` = '1997-09-22 00:00:00.0')(`order details`.`unitprice` = '12.0000')
        4 (queryID = '5')(`order details`.`unitprice` = '62.5000')(`orders`.`requireddate` = '1997-09-22 00:00:00.0')
        5 (queryID = '5')(`orders`.`requireddate` = '1997-09-22 00:00:00.0')(`order details`.`unitprice` = '46.0000')
    }
    2 (queryID = '4')(queryID = '2')(`customers`.`city` = 'Berlin')(`customers`.`customerid` = 'ALFKI')(`orders`.`orderid` = '1(
    770659098980l1_NAR_770659049036O4 {
        1 (queryID = '5')(`order details`.`unitprice` = '10.0000')(`orders`.`requireddate` = '1997-11-24 00:00:00.0')
        2 (queryID = '5')(`order details`.`unitprice` = '18.0000')(`orders`.`requireddate` = '1997-11-24 00:00:00.0')
        3 (queryID = '5')(`order details`.`unitprice` = '38.0000')(`orders`.`requireddate` = '1997-11-24 00:00:00.0')
        4 (queryID = '5')(`order details`.`unitprice` = '17.4500')(`orders`.`requireddate` = '1997-11-24 00:00:00.0')
        5 (queryID = '5')(`order details`.`unitprice` = '19.0000')(`orders`.`requireddate` = '1997-11-24 00:00:00.0')
        6 (queryID = '5')(`order details`.`unitprice` = '49.3000')(`orders`.`requireddate` = '1997-11-24 00:00:00.0')
    }
    3 (queryID = '4')(queryID = '2')(`customers`.`city` = 'Berlin')(`customers`.`customerid` = 'ALFKI')(`customers`.`contacttit
    770659127146l9_NAR_770659049036O4 {
        1 (queryID = '5')(`order details`.`unitprice` = '30.0000')(`orders`.`requireddate` = '1998-02-12 00:00:00.0')
        2 (queryID = '5')(`orders`.`requireddate` = '1998-02-12 00:00:00.0')(`order details`.`unitprice` = '12.5000')
        3 (queryID = '5')(`order details`.`unitprice` = '32.8000')(`orders`.`requireddate` = '1998-02-12 00:00:00.0')
        4 (queryID = '5')(`order details`.`unitprice` = '123.7900')(`orders`.`requireddate` = '1998-02-12 00:00:00.0')
        5 (queryID = '5')(`orders`.`requireddate` = '1998-02-12 00:00:00.0')(`order details`.`unitprice` = '25.8900')
```

Figura F.3: Amostra de coleção resultante de C_1

```
26 (queryID = '4')(queryID = '2')(queryID = '5')(`order details`.`unitprice` = '32.8000')(`customers`.`city` = 'Lule')(`emplo
27 (queryID = '4')(queryID = '2')(queryID = '5')(`customers`.`city` = 'Lule')(`employees`.`employeeid` = '1')(`orders`.`order
28 (queryID = '4')(queryID = '2')(queryID = '5')(`customers`.`city` = 'Lule')(`order details`.`unitprice` = '21.5000')(`emplo
29 (queryID = '4')(queryID = '2')(queryID = '5')(`customers`.`city` = 'Lule')(`employees`.`employeeid` = '1')(`order details`
30 (queryID = '4')(queryID = '2')(`customers`.`city` = 'Lule')(`customers`.`contacttitle` = 'Order Administrator')(`orders`.`
774032037085l3_NAR_7740318248080O {
        1 (queryID = '6')(`order details`.`unitprice` = '21.0000')(`orders`.`orderdate` = '1998-01-16 00:00:00.0')
        2 (queryID = '6')(`order details`.`unitprice` = '18.0000')(`orders`.`orderdate` = '1998-01-16 00:00:00.0')
        3 (queryID = '6')(`order details`.`unitprice` = '19.5000')(`orders`.`orderdate` = '1998-01-16 00:00:00.0')
        4 (queryID = '6')(`orders`.`orderdate` = '1998-01-16 00:00:00.0')(`order details`.`unitprice` = '34.0000')
        5 (queryID = '6')(`orders`.`orderdate` = '1998-01-16 00:00:00.0')(`order details`.`unitprice` = '33.2500')
        6 (queryID = '6')(`order details`.`unitprice` = '6.0000')(`orders`.`orderdate` = '1998-01-16 00:00:00.0')
        7 (queryID = '6')(`order details`.`unitprice` = '18.4000')(`orders`.`orderdate` = '1998-01-16 00:00:00.0')
        8 (queryID = '6')(`order details`.`unitprice` = '9.5000')(`orders`.`orderdate` = '1998-01-16 00:00:00.0')
    }
31 (queryID = '4')(queryID = '2')(queryID = '6')(`employees`.`employeeid` = '8')(`customers`.`city` = 'Lule')(`orders`.`order
32 (queryID = '4')(queryID = '2')(queryID = '6')(`employees`.`employeeid` = '8')(`customers`.`city` = 'Lule')(`orders`.`order
33 (queryID = '4')(queryID = '2')(queryID = '6')(`employees`.`employeeid` = '8')(`customers`.`city` = 'Lule')(`orders`.`order
34 (queryID = '4')(queryID = '2')(queryID = '6')(`employees`.`employeeid` = '8')(`customers`.`city` = 'Lule')(`orders`.`order
35 (queryID = '4')(queryID = '2')(queryID = '6')(`order details`.`unitprice` = '263.5000')(`customers`.`city` = 'Lule')(`orde
36 (queryID = '4')(queryID = '2')(queryID = '6')(`customers`.`city` = 'Lule')(`order details`.`unitprice` = '21.5000')(`order
37 (queryID = '4')(queryID = '2')(queryID = '6')(`customers`.`orderdate` = '1998-01-28 00:00:00.0')(`orders`.`orderid` = '10857'
38 (queryID = '4')(queryID = '2')(queryID = '6')(`orders`.`orderdate` = '1998-01-28 00:00:00.0')(`orders`.`orderid` = '10857'
39 (queryID = '4')(queryID = '2')(queryID = '6')(`orders`.`orderdate` = '1998-01-28 00:00:00.0')(`orders`.`orderid` = '10857'
```

Figura F.4: Amostra de coleção resultante de C_1 e C_2

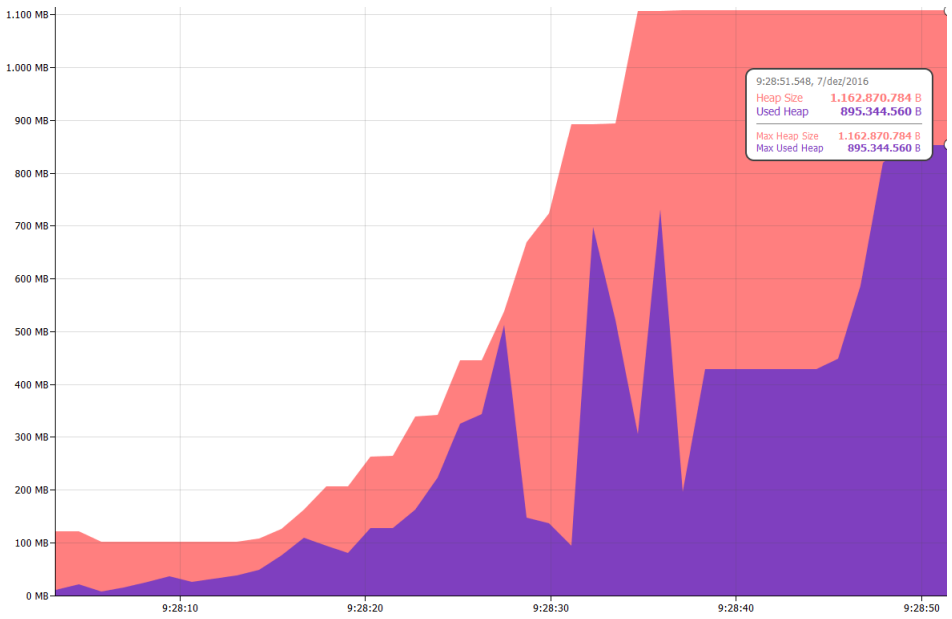

Figura F.5: Utilização de memória aquando da associação a 500 OrderID por parte do tuplo sensível

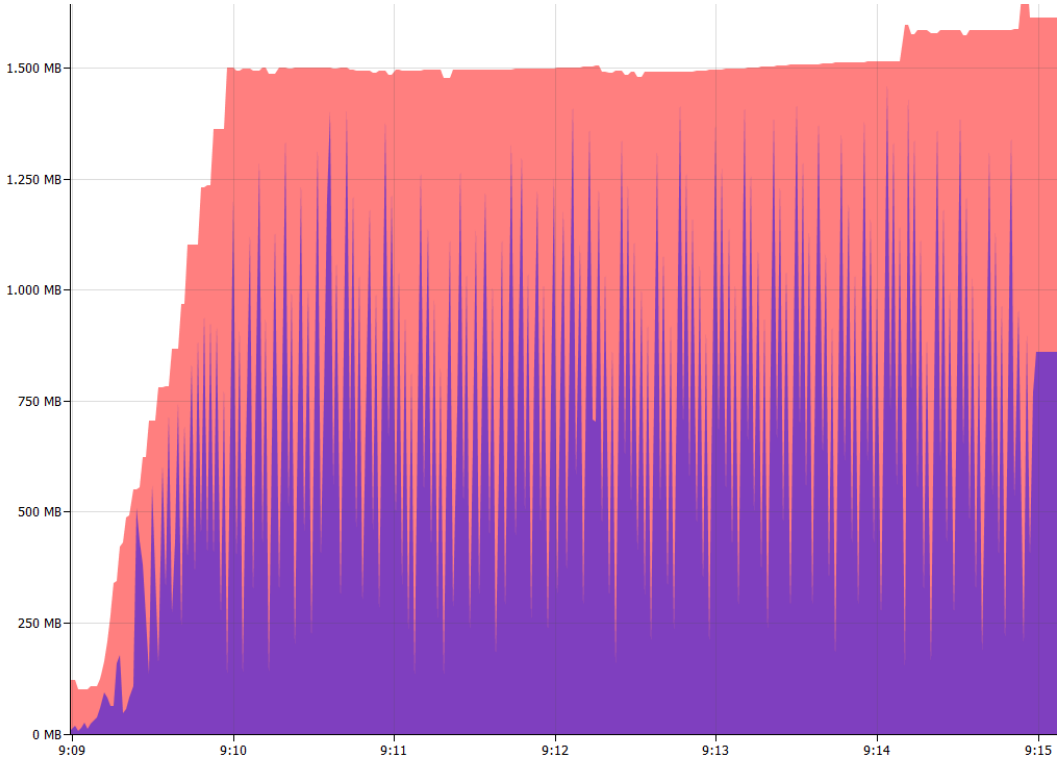

Figura F.6: Utilização de memória aquando da associação a 1000 OrderID por parte do tuplo sensível